Mining Software Repositories with CVSgrab

Lucian Voinea
Technische Universiteit Eindhoven
The Netherlands
l.voinea@tue.nl

Alexandru Telea
Technische Universiteit Eindhoven
The Netherlands
alext@win.tue.nl

ABSTRACT

In this paper we address the process and team analysis categories of the MSR Mining Challenge 2006. We use our CVSgrab tool to acquire the data and interactively visualize the evolution of ArgoUML and PostgreSQL, in order to answer three relevant questions. We conclude summarizing the strong and weak points of using CVSgrab for mining large software repositories.

Categories and Subject Descriptors

D.2.7 [**Software engineering**]: Distribution, Maintenance, and Enhancement – *documentation, reengineering*;

General Terms

Management, Measurement, Documentation

Keywords

Evolution visualization, Software visualization, CVS

1. INTRODUCTION

The MSR Mining Challenge brings together researchers and practitioners in the field of software repository mining, and stimulates them to compare their tools and approaches. To establish a common ground for comparison, two benchmarking datasets are proposed: the ArgoUML and PostgreSQL CVS repositories. ArgoUML is an open source project with a history of 6 development years, 4452 evolving files, contributed by 37 authors. PostgreSQL is an open source project with a history of 10 development years, 2829 evolving files, contributed by 27 authors. We used our CVSgrab tool [1] from the Visual Code Navigator toolset [2] to analyze the process and the team structure of these projects. The process and findings are described below.

2. SETUP

CVSgrab [1] is a tool for visualizing the evolution of large software projects. CVSgrab includes mechanisms to query CVS repositories locally or over the Internet. File contents are retrieved on demand and cached locally, which massively speeds up the mining process. CVSgrab can detect and cluster files with similar evolution patterns, using several evolution similarity metrics [1]. Unlike classical CVS clients such as WinCvs or TortoiseCVS, CVSgrab provides extensive support for interactively showing evolutions of huge projects on a single screen, with minimal browsing. Figure 1 depicts the architectural pipeline of CVSgrab:

MSR'06, May 22–23, 2006, Shanghai, China.
ACM 1-59593-085-X/06/0005.

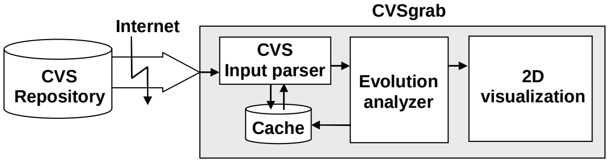

Figure 1: CVSgrab architectural pipeline

CVSgrab uses a simple 2D layout (see Figures 2,3,4): Each file is drawn as a horizontal strip, made of several segments. The *x*-axis encodes time, so each segment corresponds to a given version of its file. Color encodes version attributes, e.g. author, type, size, release, presence of a given word in the version's CVS comment, etc. Atop of color, texture may be used to indicate the presence of a specific attribute for a version. File strips can be sorted along the *y*-axis in several ways, thereby addressing various user questions.

3. RESEARCH QUESTIONS

We used CVSgrab to acquire, analyze and visualize the evolution information for ArgoUML and PostgreSQL. We formulated a number of relevant team and process related questions and tried to answer them using CVSgrab's interactive visual mechanisms:

Q1: What is / was the development process?

Assessing the development process is important for project and/or process auditors. Usually, the assessment outcome is based on developer interviews and not on the real situation. We propose using CVSgrab to base such assessments on the real data in CVS. We used CVSgrab to visually compare the development process behind both ArgoUML and PostgreSQL (Figure 2). We sorted the files in the increasing order of their creation time. We used color to encode file type: In Figure 2 left, documentation files are yellow (HTML) and light green (images) and Java sources are red. In Figure 2 right, C sources are blue, C headers are light green, test suites are red, and documentation files are green.

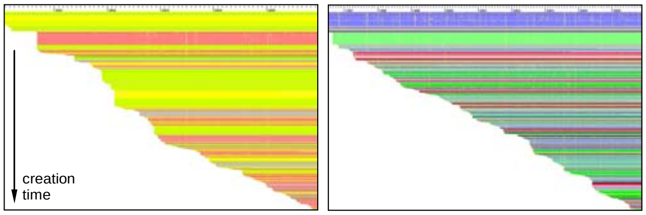

Figure 2: Evolution of file type: ArgoUML (left), PostgreSQL (right). Creation time increases from top to bottom.

We now easily see that the development of ArgoUML started with some documentation files (yellow, light green), possibly containing the system specification and/or design. Implementation source files followed only later. For a significant period, i.e. more than 1/3 of the development time, no new source files appear. This suggests the system architecture was stable in this period.

Next, source and documentation files are alternatively added in large chunks, suggesting a coarse iterative development process with few architectural changes. In contrast, the development of PostgreSQL starts directly with a set of C source files, followed shortly after by a set of header files (light green). This suggests the system, as present in CVS, was not developed from scratch, but started atop of some previous project. However, the system specification / design either does not exist, or it is not maintained: There are just a few documentation files (green) and these appear much later in the project. The system architecture appears to be less stable, as header files containing interfaces and corresponding implementation files are added throughout the entire project. The set of committed files is frequently interrupted by test suites (red). This suggests an iterative development process in which added functionality is tested before implementing new one.

Q2: What are the main contributors and their responsibilities?

During the development and maintenance of large software projects, new developers often join and/or leave the team. It is very important that newcomers quickly get familiar with the rest of the development team and their responsibilities. In Figure 3, we used the same file layout as in Figure 2 to show the evolution of the two projects. However, color encodes now the ID of the developers, so Figure 3 shows the evolution of contributions.

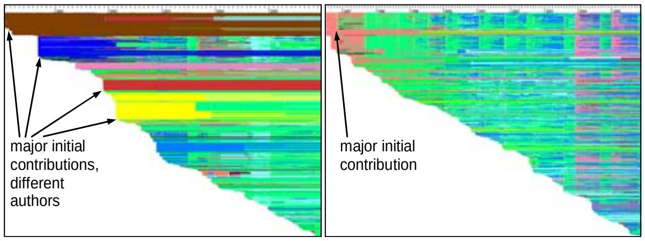

Figure 3: Evolution of author contributions: ArgoUML (left), PostgreSQL (right)

We can see that, both for ArgoUML and PostgreSQL, there is only one author for each major initial contributions, i.e. areas with a steep slope of the time curve. However, these contributions might represent the work of more developers, initially committed by one configuration manager. The evolution of PostgreSQL reveals another interesting pattern: alternative contribution of two developers, e.g. green and blue vertical stripes for the middle period of evolution. The responsibilities of the two developers are however different. We can see that the contributions of the 'green' author involve many files simultaneously, while the 'blue' author commits fewer files, but more often. This suggests the 'green' author has rather the role of a configuration manager that applies formatting changes to the entire code (e.g. indentation), while the 'blue' author affects the system functionality in small increments.

Q3: Where are located the development issues discovered and solved during alpha testing of some given release?

To track errors during debugging, it is of paramount importance to narrow down the location of the code introducing the fault. This might not always coincide with the location where the program crashes. Moreover, an error might be caused by the resolution of another issue. In such situations, it is useful to easily identify the code that changes from one system release to another *and* in the same time addresses a given issue. In Figure 4, we used the same file layout as in Figure 2 to show the evolution of ArgoUML. Color encodes versions that belong to a given system release: light green = VERSION_0_14_ALPHA_1, dark cyan = VERSION_0_14_F, red = both releases, grey = none. Grainy texture shows versions that contain a reference to the word "issue" in their associated CVS comment file. We see that only a few files that have been changed during the alpha testing of release VERSION_0_14 appear to reference the word "issue". This is shown as light green horizontal segments followed by textured dark cyan ones. At close inspection, we saw that all this code refers to ArgoUML's parsing mechanism.

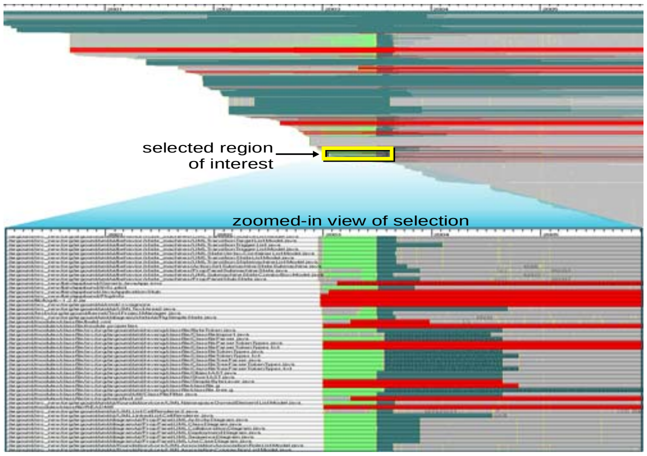

Figure 4: Identifying ArgoUML version changes between release VERSION_0_14_ALPHA_1 and release VERSION_0_14_F that contain the word "issue" in their commit comment. Inset shows a zoomed-in region, for better insight.

4. DISCUSSION

We have briefly illustrated the use of the CVSgrab tool [1] for process and team analysis of large software projects. We used as input data the MSR Challenge 2006 projects: ArgoUML and PostgreSQL. The presented use cases confirmed us that CVSgrab has a very good scalability: It can give comprehensive evolution overviews for projects of thousands of files and hundreds of versions, thus meeting industry size requirements. CVSgrab can easily answer questions that involve the formation of a large uniform color pattern, e.g. Q1 and Q2 in this paper, or questions involving comparison of a small number of colors, e.g. Q3. Secondly, the tight integration of the on-demand, Internet-based CVS data browsing, acquisition, and visualization in CVSgrab massively simplified the process of getting quick overviews of huge projects. Finally, CVSgrab can be easily extended to support different scenarios, by adding different file sorting techniques, attribute-to-color mappings, and file similarity metrics, yielding a powerful CVS mining tool. A complete version of the CVSgrab tool is available for download on the Visual Code Navigator home page at: http://www.win.tue.nl/~lvoinea/VCN.html

References

[1] Voinea, L., Telea, A. CVSgrab: Mining the History of Large Software Projects. *Proc. EUROVIS 2006*, ACM Press, 2006, to appear.

[2] Lommerse G., Nossin F., Voinea S.L., Telea A.: The Visual Code Navigator: An Interactive Toolset for Source Code Investigation. *Proc. IEEE InfoVis*, IEEE Press, 2005, 24 – 31