Visualizing Debugging Activity in Source Code Repositories

Lucian Voinea and Alexandru Telea
Department of Mathematics and Computer Science
Technische Universiteit Eindhoven, the Netherlands
l.voinea@tue.nl, alext@win.tue.nl

Abstract

We present the use of the CVSgrab visualization tool for understanding the debugging activity in the Mozilla project. We show how to display the distribution of different bug types over the project structure, locate project components which undergo heavy debugging activity, and get insight in the bug evolution in time.

1 Introduction

We consider understanding the debugging activity of the Firefox browser, part of the Open Source project Mozilla. Firefox has 659 files written by 108 authors over 4 years. The Bugzilla database for this project contains fixes for 4497 bugs from the total number of bugs reported.

We show how to use the CVSgrab visualization tool [1] to examine how the different bug types are spread across the project, which files are the most heavily influenced by bugs, and how do these correlate with authors. CVSgrab contains integrated facilities to connect to remote CVS repositories and acquire evolution data, such as file versions, commit times, authors, commit logs, and debugging data. This functionality is described in detail elsewhere [1]. The complete data acquisition process took about 45 minutes.

CVSgrab visualizes the evolution of large software projects. Figure 1 is a snapshot of the CVSgrab tool interface after we loaded the complete project. The first listbox in the GUI (Fig. 1 A) shows that, out of several available attributes (authors, file type, searched text, folders, and debugging activity), the last one was chosen for display. The second listbox (Fig. 1 B) shows the possible values for this attribute: enhancements, trivial, minor, normal, major, critical, and blocker, as reported by Bugzilla. For every value, this listbox shows a user-editable color and a metric bar with the number of occurrences of that value. We immediately see that the *normal* bugs are the highest majority (80%). The tree view (Fig. 1 C) allows classical navigation to the folder or file of interest. The main view (Fig. 1 D) shows the evolution of all files in the project. Each file is drawn as a horizontal pixel line split into segments, one per version. Segment colors show version properties, e.g. bug data, author ID, or file size. Given our attribute selection, color shows bug types. Gray indicates files with no debugging activity. Files are stacked vertically, sorted on user-defined criteria. By default, the order of files in folders is used.

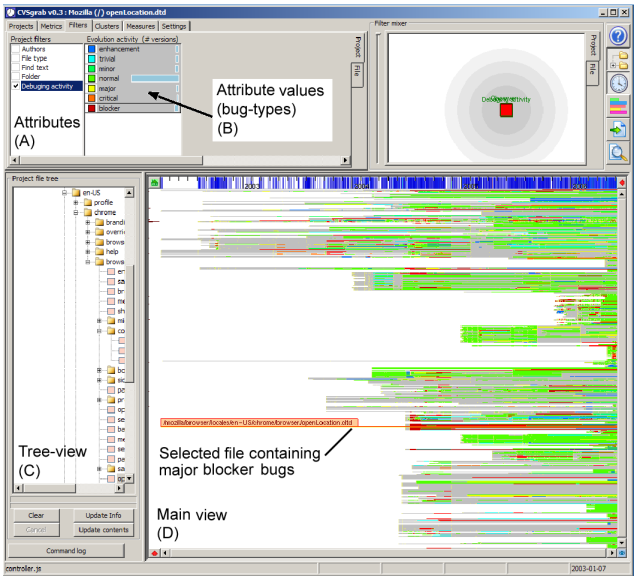

Figure 1. The CVSgrab tool showing the Mozilla project. Color shows bug types

This image already shows that most files undergo sustained debugging activity from a very early stage. This indicates a very active, dynamic project. We turn now our interest to those files drawn in red shades, i.e. showing critical or blocker bugs.

2 Showing Critical Development Areas

The first task we want to complete is to show critical development areas, i.e. files affected by critical or major bugs. For this, we first click on such a file (red pixel line) in the main view. The tool displays now its full path and name

(Fig. 1 E). That file is called */mozilla/browser/locales/en-US/chrome/browser/openLocation.dtd*. Next, we use CVSgrab's 'sort by similarity' functionality. Given a target file f_t, in our case the clicked one, and any other file f in the project, this computes the distance $d(f_t, f) \in \mathbb{R}_+$, which shows how close in time the commit moments of f_t and f are. The implementation of this technique is given in [1]. Next, we sort the files on the vertical axis in increasing distance, i.e. decreasing, similarity, order. The result (Fig. 2) shows a clear concentration of red file stripes at the top of the image (A). This is interesting, as it means that most files which evolve similarly (are committed at roughly the same moments) with our selected file, also contain major bugs. However, we see something else here. The red color appears suddenly at a given point t_0 in time, and then disappears quite quickly as we move to the right, i.e. further from that moment. This is more visible in the zoom-in view (Fig. 2 top). Looking at the timeline above the zoom-in view, we see that the time of removal is roughly six months from the appearance moment t_0. Brushing with the mouse over the red zones, we see in the commit log window that all the red areas refer to the *same* critical bug: *Merge changes from the aviary branch to start centralizing locale files (bug 250672)*. The meaning of the red stripes' length is now clear: Short stripes indicate files where the respective changes were quickly merged. Long stripes, like the one marked C in Fig. 2, indicate files where the merge happened very later, or did not happen at all, for those bars extending all the way to the right, i.e. until the current moment.

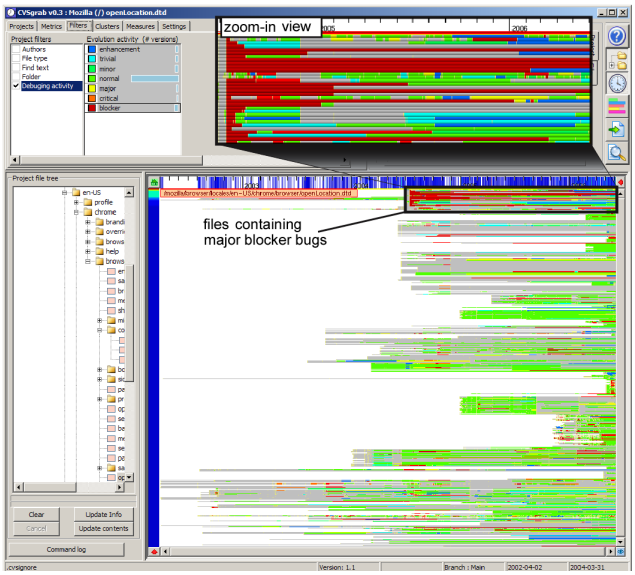

Figure 2. Files containing major bugs, sorted by similarity

The next question we have is: Are all those files with major blocker bugs close to each other in the project organization, or not? To answer this, we switch to a different color scheme. We select folders as the attribute of interest in the first lisbox (Fig. 3 A). Now every different folder in the project is shown by a different color, shown in the second listbox (Fig. 3 B). Looking at the main view, we see that the top files, which are the ones containing major blockers, have now basically two colors: brown (folder */mozilla/browser/locales/en-US/chrome/browser* and green (folder */mozilla/browser/locales/en-US/chrome/browser/bookmarks*). We conclude that there are only a few bugs marked as major blockers, localized in just two folders, a few of which have not been removed until the current date.

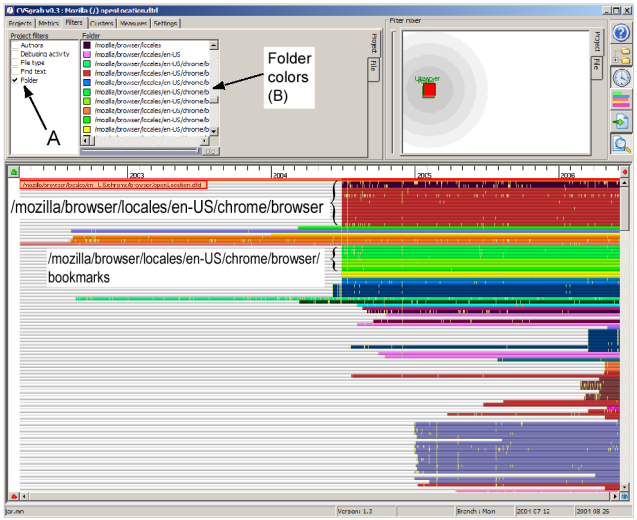

Figure 3. Folders containing major bugs

3 Conclusions

We have shown how the CVSgrab tool can be used to get insight in debugging-related activities in a large code project. CVSgrab offers only a few simple mechanisms: A file-versus-time layout, attributes shown by colors, and sorting files on various metrics. It is the orthogonal and interactive *combination* of these mechanisms that makes CVSgrab a powerful tool. The scenarios presented in this paper were done with a few (under 10) sorting and attribute selection operations, and took approximatively 10-15 minutes from the first try. CVSgrab can support many similar analysis tasks in a similar way.

CVSgrab is available at *www.win.tue.nl/∼lvoinea/VCN.html*

References

[1] L. Voinea and A. Telea. Multiscale and multivariate visualization of software evolution. In *Proc. ACM SoftVis*, pages 115–124. ACM Press, 2006.