How Do Changes in Buggy Mozilla Files Propagate?

Lucian Voinea*
Technische Universiteit Eindhoven

Alexandru Telea†
Technische Universiteit Eindhoven

Abstract

We demonstrate the use of the CVSgrab tool for assessing change propagation in the Open Source browser Firefox.

Keywords: Evolution visualization, Software visualization, CVS

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

1 Introduction

The Firefox browser, part of the Open Source project Mozilla, contains 659 files contributed by 108 authors over more than 4 years. It contains fixes for 4497 bugs from the total number of bugs reported in this period.

CVSgrab [Voinea and Telea 2006] is a tool for visualizing the evolution of large software projects. Each file is shown as a sequence of segments, one per version, aligned in time along the horizontal axis. Color shows version properties, e.g. author ID. Files are stacked vertically, ordered in user-specified ways. Atop of this layout, bug data acquired from the Bugzilla database can be shown using glyphs centered at the reported bug fix time and colored e.g. by bug severity. Files may be clustered based on evolution similarity [1] to highlight project file sets that evolve (change) together.

2 Research Question

We used CVSgrab to estimate the scope of change propagation in Firefox during debugging. The question we tried to answer was:

How do bug removal changes propagate across files?

To this end, we tried to see whether changes caused by bug removal are contained with in the folder of the file where the bug is first localized, or they ripple through the entire project.

We used the integrated data acquisition module of CVSgrab to get the Firefox evolution data from the Mozilla CVS server. This took 30 minutes. Separately from this, we used the Bugzilla web interface of the Mozilla project to acquire a report containing the up to date list of fixed bugs in the Firefox browser. Next, we included this report into the CVSgrab folder containing the Firefox evolution information. Then, we loaded the whole Firefox evolution data into CVSgrab and started the visual assessment.

First we tried to identify the project folder with the **highest density of fixed bugs**. For this, we used the *metric lists* of CVSgrab (Fig. 1). These lists are used to display and select project entities based on a given metric. For each list element, a right-aligned horizontal bar can show a metric value. The metrics available for folders are: *number of bugs* and *number of files*. Metric lists can also be sorted by users according to one of the available metrics. We sorted the folders list according to the *number of files* metric (Fig. 1b). Then we visualized the *number of bugs* metric without changing the order of the elements (Fig. 1a). One can see that the /base/content folder has the largest number of bugs, but also the largest number of files. In contrast with this, the /components/places/content folder (selected in Fig. 1) has a small number of files and yet relatively many bugs. Another possible candidate is /components/places/src. However, the ratio *number of bugs / number of files* seems to be lower than in the previous case. Hence, we finally drew the conclusion that /components/places/content is **the folder with the highest bug density** in the Firefox browser.

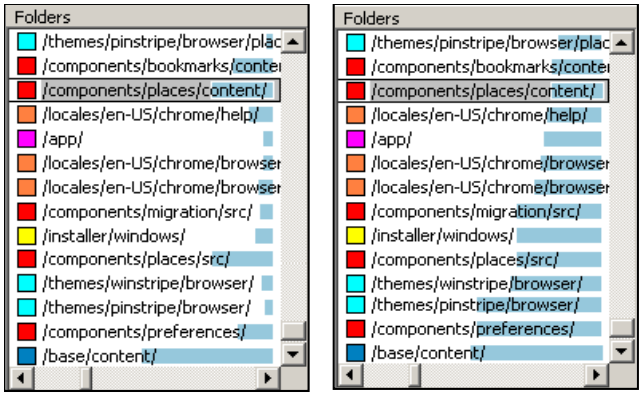

Figure 1: CVSgrab metric lists show Firefox folders sorted on number of files. Blue metric bars show the bug count (a) and file count (b) in each folder

Next, we tried to assess how changes in this folder propagate in the system. For this, we used the main evolution visualization area of CVSgrab. We used color to encode the parent folder of each file. In particular, we chose the red color for the target folder /components/places/content, and grey for the other folders.

To see how files in the target folder evolved in time, we clustered all files in the project based on *evolutionary coupling*, i.e. similarity of the files' commit moments. This method is described in detail in [Voinea and Telea 2006]. Only two clusters contained red, i.e. files from our target folder. These are clusters A and B, highlighted in Fig. 2. We tried next to identify the folders containing the other (not red) files contained in these two clusters. At a closer inspection (using the *details-on-demand* mechanism of CVSgrab) we discovered that some of these files belong to /installer/windows. We next color encoded this new folder with yellow. In the end, as depicted in Fig.2, we discovered that most of the remaining files in the two clusters belong to this folder.

* email: l.voinea@tue.nl

† email: alext@win.tue.nl

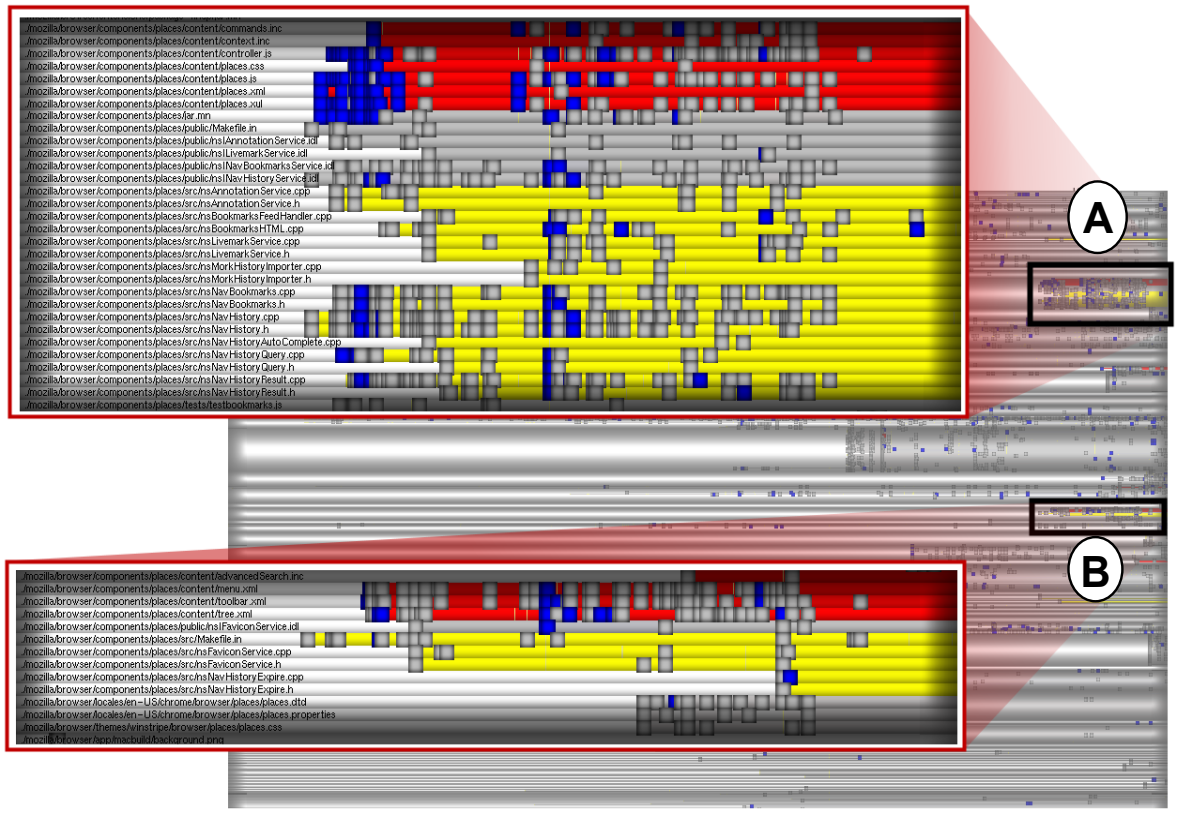

Figure 2: Buggy folder /components/places/content (red) spans two clusters together with folder /installer/windows (yellow). Critical bugs are shown as blue squares, other bugs as grey squares.

Therefore, we concluded that **there is a high change-correlation** between the folders /components/places/content and /installer/windows. This means that changes in one folder will likely propagate to the other one.

3 Discussion

Our CVSgrab visualization suggests that the folder with the highest bug density in Firefox, i.e. /components/places/content, is quite safe with respect to change propagation. Changes in this folder seem to propagate mainly to itself and to /installer/windows. Accordingly, changes as a result of a bug fix are likely to share the same propagation scope.

This type of result can be used, for example, to assess the quality of the system architecture. Assuming a folder contains the implementation of a specific structural part of a system, one can look at the bug density and change propagation to assess the architecture of that system. The most desirable situation is a system with low bug density folders, and change propagation limited to single folders (i.e. components). However, practice shows bugs can hardly be prevented. A pragmatic approach in this respect is to facilitate their removal by a highly decoupled system architecture. While this might not be affordable for the entire system, it could be worthwhile ensuring it at least for the most problematic parts. In this respect, according to our analysis, the most problematic part in the Firefox browser seems to be **relatively decoupled from the rest** of the system.

Our analysis does not include all files in the two clusters that contain the target folder. There are a few remaining files that we did not consider (drawn grey in Fig.2). Taking these files into account would improve the overall analysis results. However, the remaining files represent a relatively small percentage from the evolution clusters. Therefore, even if changes propagate to these files, the impact is less important, and propagation can still be considered as mainly localized in our two folders already found.

The complete analysis took less than 20 minutes. CVSgrab scaled very well in this case. The color encoding of folder combined with the cushion encoding of clusters made it very easy to identify the change propagation. The slowest part was the calculation of the *evolutionary coupled* clusters. However, this was performed automatically and did not require user interaction.

CVSgrab covers many other analysis scenarios as well. The latest CVSgrab tool, supporting CVS and Subversion repositories, is downloadable from the Visual Code Navigator home page at:

www.win.tue.nl/~lvoinea/VCN.html

References

VOINEA, L., TELEA, A., 2006. An Open Framework for CVS Repository Querying, Analysis and Visualization. *Proc. MSR'06*, ACM Press, 2006, pp. 33 – 39