

Scrutinize

Exploring A Project's Revision History

Chris Luce
University of Calgary
Calgary, Canada
cluce@ucalgary.ca

Jamie Starke
University of Calgary
Calgary, Canada
jrstarke@ucalgary.ca

Tom Zimmermann
University of Calgary
Calgary, Canada
zimmerth@cpsc.ucalgary.ca

Jonathan Sillito
University of Calgary
Calgary, Canada
sillito@ucalgary.ca

1. INTRODUCTION

Developing software is a highly collaborative activity involving a range of interested parties, which makes effective coordination and progress tracking both important and difficult. This coordination centers on the source code for a project which is commonly managed using a version control system. A version control system archives and makes available information about each change made over the lifetime of a code base, such as the author of the change and the affected source code lines.

Despite recent research demonstrating the value of the archived information (e.g., [1, 7]), coordinating and tracking the progress of projects within an organization (both co-located and distributed) remains difficult [5]. We believe that archived information is underutilized partially because it is stored separately in various systems (making cross-tool inference difficult), and also because it is generally worked with and viewed at a low level (making trends or other patterns difficult to observe).

Scrutinize is a new web based tool we have developed as part of a larger project to empirically investigate how project archives can be more effectively leveraged to provide various stakeholders with relevant and timely information. Scrutinize allows users to explore revision information for a project (specifically revision information from Subversion¹, a popular open source version control system). Our video demonstrates the main features of the tool and shows how it can be used to learn about what changes have been made and who has been making them.

2. RELATED WORK

In developing Scrutinize, and related tools, we are building on previous work that presents visual project information

¹<http://subversion.tigris.org>

and supports interaction with that information. Froehlich and Dourish [3], for example, present a tool that provides a visual representation of artifacts and activities and supports the exploration of relationships between those. Their focus is on what is happening currently in a project and the information presented is at the source code line level (i.e., at a low level), while Scrutinize also supports the exploration of history and operates at the author and module level.

Other work has proposed visualizations of project data to show software structure, including using the structure to support coordination [6], show relationships between artifacts and activities [3], and to manage change requests [4, 2]. Scrutinize does not make use of structural information and does not currently incorporate change request information. The relationships shown by Scrutinize are between module and author, and between module and module for modules that tend to be modified together.

3. SCRUTINIZE

Scrutinize is a web based tool designed to take information from a source code repository, and present it in a way that allows project team members to learn about how the project has been changing and who has made those changes.

As shown in Figure 1, Scrutinize's interface is divided into 4 panels: a timeline (see part A of the figure), an authors list (part B), a module list (part C) and a commits list (part D). The timeline shows the number of commits to the project source code repository over time. The author list lists everyone that has committed to the project, with indicators showing the relative number of commits by each author. The module list lists each of the modules in the code base along with the indicators showing the relative number of times each module has been changed. Commit details are shown in the commits list.

When the user hovers over an author, the modules and commits associated with that author are highlighted. Similar highlighting occurs when hovering over a module, or a commit. For example, the highlighting in Figure 1 shows that the given commit was made by author *mtredinnick* and affected four different modules.

The number of commits in a project can be quite large, so Scrutinize provides a number of ways for a user to specify which commits she is interested in. The grey bars on either side of the timeline can be dragged left or right to temporally

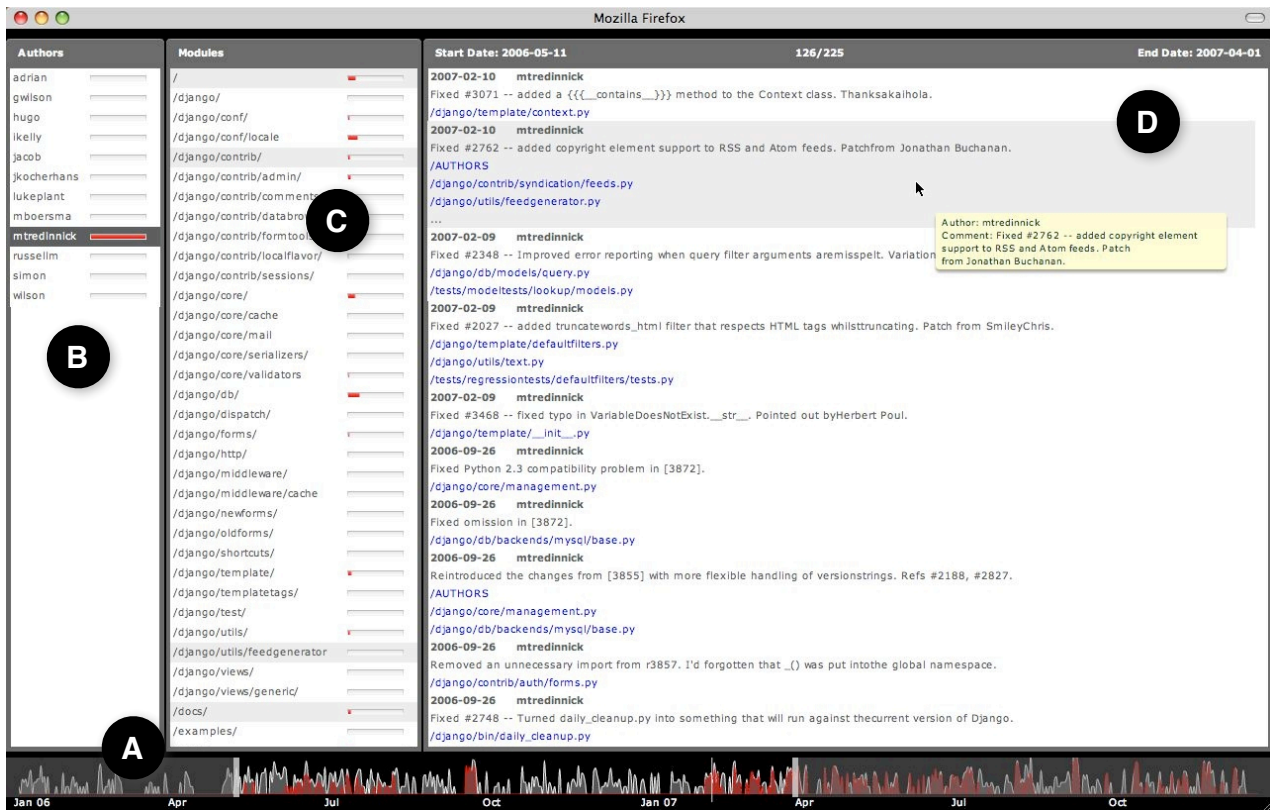


Figure 1: A screenshot of the Scrutinize Tool.

scope the list. When an author is selected the tool will only present the commits made by that author. Similarly, when a module is selected, the tool only presents the commits made to that module. When the commits are filtered in this way a red overlay graph in the timeline shows the number of commits for that author and/or module.

4. VIDEO CONTENTS

The data presented in the video is from the Django open source project² which is publicly accessible. To prepare for the video we downloaded two years of data from the project's Subversion repository and loaded that into an SQL database.

The first part of our video presents the main features of Scrutinize. The last part of the video works through two short examples of how the tool can be used. The first example shows how the tool can be used to explore the work done by a particular author. The second example shows how the tool can be used to explore who has been making changes to a particular module and what other modules are generally changed with that module.

5. REFERENCES

[1] D. Cubranic, G. C. Murphy, J. Singer, and K. Booth. Learning from a project history: A case in software development. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work*, pages 82–91, 2004.

[2] J. B. Ellis, S. Wahid, C. Danis, and W. A. Kellogg. Task and social visualization in software development: Evaluation of a prototype. In *Proceedings of CHI*, pages 577–586, 2007.

[3] J. Froehlich and P. Dourish. Unifying artifacts and activities in a visual tool for distributed software development teams. In *Proceedings of the International Conference on Software Engineering*, pages 387–396, 2004.

[4] C. A. Halverson, J. B. Ellis, C. Danis, and W. A. Kellogg. Designing task visualizations to support the coordination of work in software development. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work*, pages 39–48, 2006.

[5] J. D. Herbsleb and A. Mockus. An empirical study of speed and communication in globally distributed software development. *IEEE Transactions on Software Engineering*, 29(6):481–494, 2003.

[6] C. O'Reilly, D. Bustard, and P. Morrow. The war room command console: Shared visualizations for inclusive team coordination. In *Proceedings of the Symposium on Software Visualization*, pages 57–65, 2005.

[7] T. Zimmermann, A. Zeller, P. Weissgerber, and S. Diehl. Mining version histories to guide software changes. *IEEE Transactions on Software Engineering*, 31(6):429–445, 2005.

²<http://djangoproject.com>