A Demonstration of AZURITE: Backtracking Tool for Programmers

YoungSeok Yoon
Institute for Software Research
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213, USA
youngseok@cs.cmu.edu

Brad A. Myers
Human-Computer Interaction Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213, USA
bam@cs.cmu.edu

***Abstract*—Programmers often need to backtrack, but backtracking support in modern programming environments is limited. Previously, we have conducted a series of studies, which discovered that backtracking in programming is in fact prevalent and programmers need better backtracking tools. In this demonstration, we will present our backtracking tool called AZURITE, which provides selective undo and history search and visualization features in the Eclipse code editor. The demonstration will include the user interface presented in our previous work, as well as the new features added in response to user feedback.**

Keywords—selective undo; backtracking

I. INTRODUCTION

Programming involves both forward and backward edits on source code. Programmers not only need to write new pieces of code, but also need to revert some code to an early version for various reasons. For example, programmers make mistakes while writing code that they need to revert, or they may intentionally make temporary changes to the code, either as an experiment or to help with debugging. We call the behavior of reverting some code to an earlier version *backtracking* [1].

We conducted a series of studies about the backtracking practices of programmers. First, we conducted a lab study and a survey, which confirmed that backtracking is in fact prevalent and programmers often have problems when they want to backtrack [1]. As a follow-up, we recently conducted a longitudinal study of backtracking, which analyzed 1,460 hours of fine-grained code editing history of 21 programmers. The results showed that these programmers backtracked 10.3 times per hour on average, and 34% of all the detected backtracking instances were performed manually without using the undo command or any other tool support [2].

In fact, backtracking support is limited in modern programming environments. For example, the restricted linear undo model, which is widely used in most text and code editors, is limited in that it can only revert the most recent changes, and loses all the undone commands when the user makes new changes after undoing. Other support comes from using a version control system (VCS) such as Subversion or Git to revert some code to a previous version. However, this approach relies on the assumption that the desired code is already committed to the VCS, which may not be the case.

These problems can be resolved by having a selective undo feature in the code editor. Users could select specific edit operations performed in the past, for example the insertions of various print statements for debugging, and invoke the selective undo command to revert only the code affected by those operations. To demonstrate the feasibility of this approach, we built a tool called AZURITE (Adding Zest to Undoing and Restoring Improves Textual Exploration), an Eclipse plug-in that helps programmers to perform various backtracking tasks with selective undo.

In this demonstration, we will present our AZURITE and show how selective undo can help programmers to backtrack effectively. The demonstrated features will include the user interfaces that were previously published [3], as well as newly added features designed based on user feedback (Section III).

II. USER INTERFACES OF AZURITE

A. Timeline Visualization

In our previous paper, we presented the initial user interface of AZURITE which visualize the fine-grained code edit history, and provide the selective undo feature of the code editor. The timeline visualization is the basic user interface used to interact with the edit history. More details of timeline visualization and the other user interfaces can be found in [3].

The timeline visualization lets users see and interact with the edit history (Fig. 1 shows the updated version of the timeline). The horizontal axis represents time, and the edit history of each file is shown in a corresponding row. Each edit operation is represented as a rectangle, which is color-coded according to the type of edit: inserts are green, deletes are red, and replacements are blue. Users can click or drag to select one or more rectangles. Once some operations are selected, users can invoke a popup context menu containing various commands including selective undo.

B. Improvements to the Timeline View

We made some significant improvements to the timeline visualization (Fig. 1). One of the common ways of backtracking is to go back to a certain point in the past when a specific event happened. As Beck says in his book, “it would be great if the programming environment helped me with this, working as a checkpoint for the code every time all of the tests

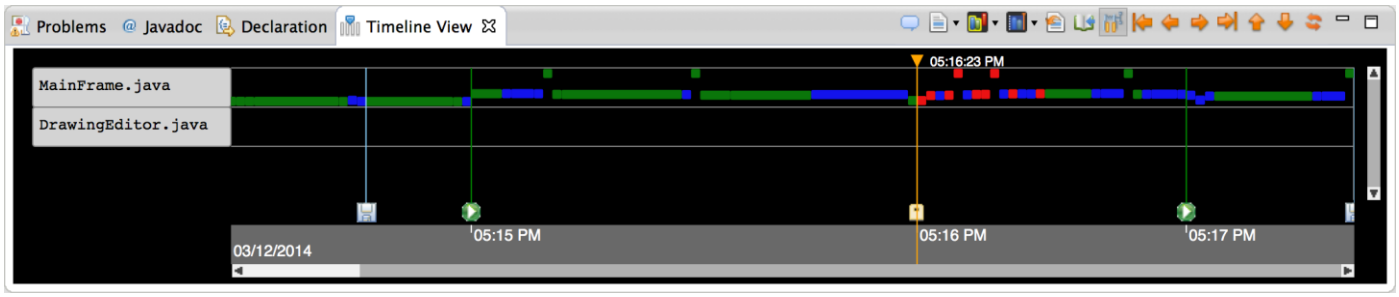

Fig. 1. The updated timeline visualization with automatically captured significant coding event icons shown just above the timecodes on the bottom. The displayed events from left to right are (1) file save, (2) run/debug application, (3) a user-defined tag, (4) run/debug application again, and (5) file save.

run" [4]. To support this, AZURITE detects significant coding events and displays them in the timeline view. Currently, the displayed events are running JUnit tests which pass ( ) or fail ( ), running the application under development ( ), saving files ( ), and version-control system related commands such as commit ( ). An event is displayed on the timeline as a vertical line with an icon representing that event at the bottom (Fig. 1). Further event types can be trivially added in the future.

Users can also "tag" the current point in time, so that it can be easily referred back to when needed, which was one of the most requested features from our preliminary field study participants. A tag works exactly same as the other events, and shows a tag-shaped icon (, Fig. 1 at 5:16pm). Users can give a name to each tag (shown on mouse hover), or leave it anonymous.

Users can left-click on any event or tag icon to move the time marker (shown in orange) to that location to see how the code looked at that point. Right-clicking any of the icons brings up a context menu with convenient commands such as "undo all files to this point," which can be viewed as a lightweight, automatic versioning feature.

C. *Interactive Selective Undo Dialog*

We designed another user interface called the *interactive selective undo* dialog in response to feedback from our preliminary user studies. The design was inspired by Eclipse's refactoring wizard, which shows a preview of all changes to be made before actually changing the code. Similar to a typical refactoring wizard, our interactive selective undo dialog shows a side-by-side "diff" view where the left panel shows the current code and the right panel shows the preview of the selective undo with the currently selected rectangles in the timeline. On the top panel is the list of all the affected files and chunks within each file in a tree form.

The interactive selective undo dialog is modeless, and the rectangles can be added to and/or removed from the selection while the dialog is open, which immediately updates the preview result shown in the dialog. By allowing this, users need not worry about selecting the exact set of rectangles on their first attempt. Users can manipulate the selection until the preview shows the desired result.

Additionally, the interactive selective undo dialog offers a convenient way for users to "keep some code unchanged" when performing selective undo. Users can select an arbitrary region of the code in the left panel, right-click to bring up the context menu, and select "Keep this code unchanged". Doing this searches for all operations affecting that selected region of code and excludes those operations from what will be undone. This feature provides a significant usability improvement, because users can easily get the desired results by roughly over-specifying the selected operations, and then marking all the code fragments desired to be in the resulting code.

D. *Regional Undo Shortcut*

The interactive selective undo dialog is the most flexible and powerful user interface we provide. However, the complexity of a dialog interface might incur unnecessary overhead for relatively simple tasks. We have found that the most popular form of selective undo is reverting a specific region of code to an old version, which has been referred to as "regional undo" [5]. In the current version of Azurite, users can select some region in the regular code editor and use a keyboard shortcut (Ctrl+Alt+Shift+Z by default) one or more times to perform selective undo on that region directly within the code editor.

III. CONCLUSION

Although selective undo can be a powerful way to backtrack, this idea has not been implemented in any popular code editors, due to the specific challenges of selective undo for text editing. We present a novel selective undo tool called Azurite for helping programmers to backtrack more easily. We expect that demonstrating this tool would generate constructive discussion on the usefulness of the tool and ways to improve the existing user interfaces. AZURITE is available for general use as an Eclipse plugin (http://www.cs.cmu.edu/~azurite/).

REFERENCES

[1] Y. Yoon and B. A. Myers, "An Exploratory Study of Backtracking Strategies Used by Developers," *Proc. CHASE 2012*, pp. 138-144.

[2] Y. Yoon and B. A. Myers, "A Longitudinal Study of Programmers' Backtracking," *Proc. VL/HCC 2014*, to appear.

[3] Y. Yoon, B. A. Myers, and S. Koo, "Visualization of Fine-Grained Code Change History," *Proc. VL/HCC 2013*, pp. 119-126.

[4] K. Beck, *Test-Driven Development: By Example*, Addison-Wesley Professional, 2002.

[5] R. Li and D. Li, "A Regional Undo Mechanism for Text Editing," *Proc. IWCES 2003*.