Better Backtracking Support for Programmers

YoungSeok Yoon
Institute for Software Research
Carnegie Mellon University
Pittsburgh, PA, USA
youngseok@cs.cmu.edu

***Abstract*—Programmers often need to backtrack while coding, yet there is only limited support for backtracking in modern programming tools. Our study results confirmed the prevalence of backtracking and identified several problems programmers face while backtracking. To mitigate these problems, we are building an IDE plug-in aimed at providing better support for backtracking by combining a selective undo mechanism, novel visualizations, and code change history search features. We envision that this approach will help programmers perform backtracking tasks more easily.**

Keywords—backtracking, selective undo, history search, integrated development environments, software visualization

I. INTRODUCTION

When developing software, programmers often need to backtrack while implementing or debugging a feature. There are various reasons for backtracking. For example, when a feature does not work as imagined, the programmer would have to revert some of the newly made code changes and try out something else. In addition, a programmer might intentionally explore different options when there are alternative solutions to a given problem or when learning an unfamiliar API. Another example situation would be adding print statements in multiple locations for debugging purposes and then removing them after successfully fixing the bug.

However, modern programming environments provide only limited support for backtracking. The linear undo command used by most existing code editors can be used in some backtracking situations. However, linear undo can only undo the most recent changes, and loses the undone changes once the programmer makes some new changes after invoking the undo commands. Another option is to use a version control system (VCS) to revert some code to a previous version, but this approach relies on an assumption that the desired code is already committed to the VCS, which may not be the case. Moreover, neither of these approaches helps when there are some wanted and unwanted code changes intermixed in the recent history.

This paper gives an overview of my on-going research that tries to address these problems with backtracking. Section II briefly reviews the exploratory studies on backtracking, and Section III describes the prototype tool called AZURITE. Then the current limitations of our approach and our future work are presented in Section IV.

II. EXPLORATORY STUDY

We performed an exploratory study of backtracking in order to get insights on what kinds of backtracking situations programmers face, what problems they have while backtracking, and how they resolve the problems using existing environments [1]. A lab study with 12 professional developers revealed that programmers have difficulty in finding relevant sections of code to backtrack. Also, programmers often commented out code just in case it is needed later, but still they lost some code fragments that turned out to be needed. Our follow-up survey confirmed that most programmers reported they need to backtrack in various situations.

III. APPROACH

The goal of this research is to build a better tool that helps programmers to perform various backtracking tasks in more natural ways. To achieve this goal, we are building a prototype tool called AZURITE (www.cs.cmu.edu/~azurite/) as an Eclipse plug-in. Our key insight is that many of these problems can be solved by having a *selective undo* feature in the code editor.

A. Selective Undo Mechanism for Code Editors

Selective undo has been well studied for creation-oriented graphical applications [2, 3], but it has not been used with text/code editors due to the following challenges. First, unlike graphical applications, text just has a stream of characters without the notion of identifiable objects. Second, there are many *conflicts* among the edit operations that can occur when the region of a new edit overlaps the region of some earlier performed edit.

We developed a selective undo algorithm for code editors [4] which can handle these issues. The tool keeps track of the segment (offset and length) information of individual fine-grain edits and updates the information as necessary. It also detects conflicts as they occur, to provide reasonable options when the user tries to selectively undo an edit which has conflicts. Another merit of our selective undo mechanism is that the user can select *multiple* edits and undo them all at once. This is not only convenient for the users, but also has a significant merit over undoing one edit at a time, because the tool can always perform undo correctly when the conflicting edits are being undone together.

It is also important to provide natural user interfaces for selective undo, which is a challenging task. Most existing selective undo user interfaces for graphical applications present

a list of operations performed in the past so that users can choose the right operation to be undone. In contrast, text editing operations are often quite fine-grained so it is hard for users to interpret the high level edit intent just by looking at the individual edits. To address these issues, AZURITE provides the following interfaces: a timeline visualization, a code history diff view, and a history search dialog.

B. Timeline Visualization & Code History Diff View

Instead of having a textual list of edits, AZURITE provides a timeline-based two dimensional visualization of the code edit history. The horizontal axis represents time, and each row represents each file being edited. Individual edits are represented as color-coded rectangles, where the vertical location in the row represents where in the file the edit happened. Users can select those rectangles to invoke various editor commands including selective undo.

Another interface is code history diff view. Users can select an arbitrary region of code and launch this view to see the fine-grained code change history of the selected region. Users can move between different versions of this code, and even revert the code to one of the previous versions, which can be considered as a specialized form of selective undo [5].

C. History Search

Once the history gets larger, it would be more difficult for users to find relevant edits to undo, which suggests that better user interfaces are needed to mitigate this problem. One of the observations from the previous lab study is that programmers remember some characteristics of the code that they want to backtrack. Examples of these characteristics include the relevant code element names (e.g., variable name, class name), location of the code, when the changes were made, etc. Along this line, AZURITE provides a *history search* feature which enables users to specify various options to search for code changes in the history. History search would also reduce the chance of having irresolvable conflicts, because the conceptually related edits are likely to be selected together with a history search query, and thus they would usually have conflicts only among themselves, which AZURITE handles automatically.

Currently, the tool provides three search options, which can be combined to form a more specific query. Users can search for all edits performed:

- on a specific region of code
- during a time where a specific text existed in the code
- within the current editing session or any time.

The search results are displayed in the timeline, and the user can further investigate the selection or invoke selective undo on the corresponding rectangles.

IV. LIMITATIONS AND FUTURE WORK

Although we have already implemented some history search options, we plan to implement more. The goal is to provide a rich set of search options so that users can express what they remember about the code changes that they want to revert in a natural way. In order to do this, we should first know about the things the users remember about the code changes.

Since our tool has many user interfaces, it is likely that there are many as yet undiscovered usability problems. We will perform a series of usability evaluation studies to improve the tool iteratively. Along the same lines, the actual effectiveness of AZURITE has not yet been shown by formal user studies. We are performing a field study with real users to see whether AZURITE helps their actual daily development activities.

We have also been collecting programmers' code editing data at a fine-grained level. Currently, our data contain more than 927 hours of coding activities collected from 9 programmers who did their own work while the logging plug-in was running. These data are not yet fully analyzed. We plan to investigate the backtracking incidents in this large dataset and extract more insights which would eventually guide us to improve our tool. Investigating how those backtracking tasks could have been achieved if they had AZURITE is another method of evaluation. In addition, there are many minor questions that could potentially be answered by analyzing the log data. For example, how many files should be shown at a time in the timeline to make it the most useful? Would it be enough to keep the current session's history for most cases, or is more past history needed in general? By finding answers to these questions from the data, user interfaces could be provided that are comfortable to use while minimizing the effort needed for future lab user studies.

V. CONSLUSION

We believe that a usable selective undo tool integrated with the code editor would help programmers perform their daily backtracking tasks more easily. In addition, this would make programmers more comfortable to explore, because they know that they can revert incorrect changes at any time, which hopefully will, in turn, enable developers be more creative and productive.

ACKNOWLEDGMENT

Funding for this research comes in part from the Korea Foundation for Advanced Studies (KFAS) and in part from NSF grant IIS-1116724. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect those of KFAS or the National Science Foundation.

REFERENCES

[1] Y. Yoon and B. A. Myers, "An exploratory study of backtracking strategies used by developers," Proc. International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE 2012), 2012, pp. 138-144.

[2] T. Berlage, "A selective undo mechanism for graphical user interfaces based on command objects," ACM Transactions on Computer-Human Interaction, vol. 1, 1994, pp. 269-294.

[3] B. A. Myers and D. S. Kosbie, "Reusable hierarchical command objects," Proc. SIGCHI conference on Human factors in computing systems: common ground (CHI 1996), 1996, pp. 260-267.

[4] Y. Yoon, B. A. Myers, and S. Koo, "Visualization of Fine-Grained Code Change History," accepted for publication at VL/HCC 2013, in press.

[5] R. Li and D. Li, "A regional undo mechanism for text editing," Proc. International Workshop on Collaborative Editing Systems (IWCES 2003), 2003.