Our Tool: Navigating Program Flow in the IDE

Chris Brown, Justin Smith, Tyler Albert, and Emerson Murphy-Hill
Department of Computer Science
North Carolina State University
Raliegh, North Carolina 27606
Email: {dcbrow10, jssmit11, tralber2}@ncsu.edu, emerson@csc.ncsu.edu

***Abstract*—Program navigation is a critical task for software developers. Unfortunately, the current state-of-the-art tools do not adequately support developers in simultaneously navigating both control flow and data flow (i.e. program flow). To assist developers in effectively navigating program flow we designed and implemented a tool that leverages powerful program analysis techniques while maintaining low barriers to invocation. Our tool enables developers to systematically navigate program flow upstream and downstream within the Eclipse IDE. Based on a preliminary evaluation, our tool is awesome!**

I. Introduction

Modern software systems contain millions of lines of source code. As software grows in size and complexity, developers increasingly rely on tools to help them navigate the programs they create. Program navigation is a central task tied to many critical activities, including exploring new code bases, debugging, and assessing security vulnerabilities.

While navigating programs, developers ask questions about control flow and data flow throughout the program [2], [3]. We will refer to these two concepts together as *program flow*. Developers are interested in navigating program flow to trace how data is modified across multiple method invocations.

Integrated development environments (IDEs) present code linearly in the order methods are defined. However, successful developers do not navigate source code linearly (line by line starting at the top of the file). Instead, they methodically navigate the code's hierarchical semantic structures [1]. To resovle this conflict and realize their ideal navigation strategies, developers rely on program navigation tools.

In this work we will design, implement, and evaluate a program navigation tool. To address the limitations of existing program navigation tools, our tool will embody five key design principles (Section II). We will implement our tool as a plugin to the Eclipse IDE.

II. Design Principles

In this section we describe the design principles that we used to shape our tool. We derived these design principles by evaluating the limitations of existing program navigation tools.

A. Powerful Program Analysis

By leveraging powerful program analysis techniques, navigation tools can provide more accurate information. For example, by analyzing abstract syntax trees (ASTs) and call graphs, tools can make proper references to variables and methods. Simple textual analysis may lead to inaccurate results, especially when programs include inheritance and duplicate variable names.

B. Low Barriers to Invocation

Barriers to invocation may inhibit adoption. As developers may wish to navigate multiple program paths concurrently, repetitively invoking the tool may be cumbersome, especially if barriers are high.

C. Full Program Navigation

Developers are not only interested in traversing programs' call graph, but also how data flows through the call graph. To do so, developers must inspect the relationship between methods as well as the methods themselves. Often the methods of interest span across multiple source files. Furthermore, program navigation tools should support this traversal both upstream and downstream. That is, tools should highlight variable assignments and also subsequent variable uses.

D. Enables Systematic Evaluation

Program navigation tools should help developers keep track of where they have been and where they are going. Especially while attempting to resolve complex defects, developers may want to thoroughly explore all program paths. Their navigation tools should help them keep track of their progress.

E. In Situ Results

Switching between views in the IDE can cause disorientation [Cite]. As developers navigate through code, navigation tools should present their results in that context. When navigation tools present results outside the code, developers are burdened with the cognitive load of translating those results back to the code.

III. Related Work

Summary of related work, including a table evaluating existing tools on various design principles. Spoiler alert, none of the tools satisfy all of the principles.

IV. [Name of Tool]

We implemented our tool as a plugin to the Eclipse IDE [cite]. We chose Eclipse because of its popularity and extensibility. Eclipse is one of the most widely used open source IDEs for Java development and it provides many extension points for plugins.

V. Preliminary Evaluation

We plan to perform a preliminary evaluation of our tool with approximately five to ten developers. Our goals are to get some initial feedback on the usability of our tool and evaluate whether our approach shows promise in enabling developers to efficiently navigate program flow.

We will ask developers to perform two tasks that involve program flow navigation (Task 1 and Task 3 from [3]). We will measure their accuracy, speed, and administer a post-test usability questionnaire.

VI. Results

Hopefully, participants are faster/more accurate with the tool. Rate positive experience on the questionnaire.

VII. Discussion

VIII. Limitations

IX. Conclusion

Acknowledgment

The authors would like to thank...

References

[1] M. P. Robillard, W. Coelho, and G. C. Murphy, "How effective developers investigate source code: An exploratory study," *IEEE Transactions on Software Engineering*, vol. 30, no. 12, pp. 889–903, 2004.

[2] T. D. LaToza and B. A. Myers, "Hard-to-answer questions about code," in *Evaluation and Usability of Programming Languages and Tools*. ACM, 2010, p. 8.

[3] J. Smith, B. Johnson, E. Murphy-Hill, B. Chu, and H. R. Lipford, "Questions developers ask while diagnosing potential security vulnerabilities with static analysis," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2015. ACM, 2015, pp. 248–259. [Online]. Available: http://doi.acm.org/10.1145/2786805.2786812