Focusing on Execution Traces Using Diver

Del Myers
Department of Computer Science
University of Victoria
Victoria, British Columbia, Canada
Email: delmyers.cs@gmail.com

Margaret-Anne Storey
Department of Computer Science
University of Victoria
Victoria, British Columbia, Canada
Email: mstorey@uvic.ca

***Abstract*—Understanding the behaviour of software is an essential part of program understanding in general. Execution traces of running software can be used as a source of information about software behaviour. Unfortunately, execution traces tend to be extremely large, making it difficult to show users the information that they need. This demo presents new developments in our tool called *Dynamic Interactive Views for Reverse Engineering* (Diver), which attempts to address this difficulty.**

***Index Terms*—Software engineering, reverse engineering, graphical environments, integrated environments**

I. Introduction

During software maintenance activities, a significant amount of time and effort is invested in program understanding tasks [1]. The static and dynamic behaviour patterns of software are both important components of program understanding [2]. Some understanding of dynamic behaviour can be elicited by manually walking through source code, but this is time consuming and can be incomplete due to programming language features such as dynamic binding.

Alternatively, maintainers can record execution traces of running software. Such traces faithfully represent the runtime behaviour of software, but they tend to be extremely large. Tools are necessary to aid users in their investigation of execution traces and creating such tools is a significant challenge. If care is not taken, there is serious risk of actually increasing a maintainer's work load as they are tasked with investigating not only a large code base, but also large amounts of execution trace data.

This demo presents a tool called *Dynamic Interactive Views for Reverse Engineering* (Diver), which we created to aid software developers in their program understanding tasks. It is built on the Eclipse IDE[1] and is designed to transform the familiar IDE into a tool for program understanding based on dynamic execution traces of Java software. Diver does this through a technique we call the *trace-focused user interface*. While Diver was briefly introduced in another short tool demo [3], it now has a fuller integration with the Eclipse IDE and better filtering in its primary visualization (sequence diagrams). This demo features these recent developments.

II. The Trace-Focused User Interface

Research indicates that developers' productivity increases if their IDEs highlight the artifacts that are pertinent to the task at hand [4]. Execution traces contain information about what is pertinent to program behaviour. The challenge is to extract the interesting information and present it in a useful way.

Diver uses a method called software reconnaissance [5]. Software developers can use Diver to gather several execution traces of the software under investigation, some that exhibit a feature of interest, and some that do not. These traces are displayed in a custom view called the *Program Traces* view. In this view, users can "activate" a trace that exhibits the feature of interest, and "hide" traces that do not (Fig. 1). Activated traces are displayed using a green indicator icon, and an "open eye." Hidden traces show a "closed eye." Software reconnaissance is used to discover what artifacts are unique to the feature of interest. This is the essence of the trace-focused user interface. Diver uses this information to filter the standard Eclipse *Package Explorer* view so that it displays only the classes and methods that are unique to the activated trace.

Fig. 1. The Program Traces View

.

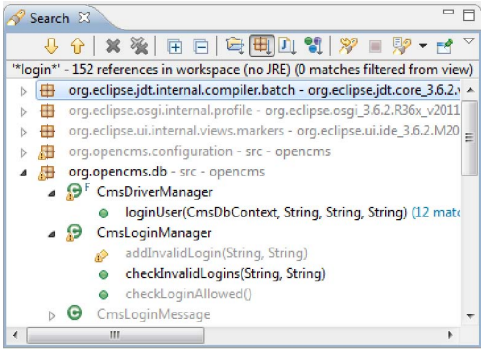
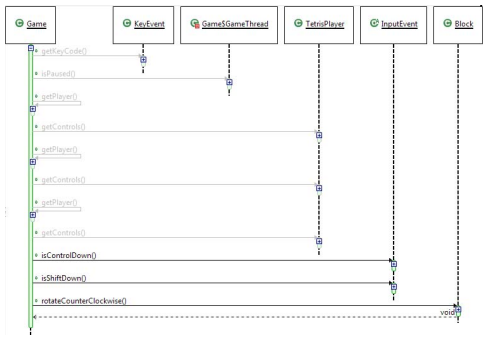


Fig. 2. Java Search Results, Highlighted by the Trace-Focused User Interface

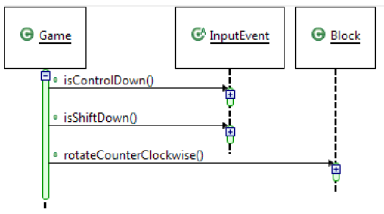
New to Diver is the ability to affect any view that displays Java code elements, including the source code editor, the

[1]http://www.eclipse.org

1095-1350/11 $26.00 © 2011 IEEE
DOI 10.1109/WCRE.2011.66



(a) Greying-out method executions



(b) Hiding method executions

Fig. 3. Using Software Reconnaissance in Sequence Diagrams

Hierarchy view, the *Outline* view, and the *Search Results* view. For example, the trace-focused user interface can be used to refine the results of a Java code search. Developers have difficulty with code searches because their keywords often return too many results [6]. Diver annotates the Eclipse *Search Results* view to highlight the results that are unique in the activated trace. Fig. 2 shows the results of a Java Search for the log-in process of an Open Source web application. The original search returned 152 results, but according to Diver, only 11 of them are relevant to the "log-in" feature (captured using execution traces). The relevant results are displayed as darker text, and they can be cross-referenced with the filtered Package Explorer to help programmers gain insights about the specific details of their software.

III. Using Software Reconnaissance to Filter Sequence Diagrams

Diver includes an interactive sequence diagram viewer, which can be used to visualize a thread of execution by opening it from Diver's *Program Traces* view. Sequence diagrams are a common way to visualize program execution traces, but they tend to be extremely large, even for simple programs, which can make them difficult to understand.

Previous versions of Diver attempted to address this problem in several ways. First, we developed a method of compacting the view by using loops found in source code [7]. Second, users could right-click on Java elements in the filtered Package Explorer and use them as an aid to navigation into the sequence diagram. For example, right-clicking on a method offers the user a context menu action that will display the first call to that method in a sequence diagram.

These versions of Diver, however, did not apply any filters to the sequence diagram itself. A user was able to see and traverse a call tree in the diagram even if it was not related to a feature of interest (as described using software reconnaissance). We have recently developed a way that can efficiently apply software reconnaissance to large sequence diagrams.

Diver users can choose to either grey-out method executions that are not related to the feature of interest (Fig. 3(a)), or they can remove them completely (Fig. 3(b)). This filter can be used to prevent users from viewing parts of the sequence diagram that are irrelevant to their current understanding task. Our preliminary investigation into this technique indicates that it can reduce the size of many sequence diagrams by 80%.

IV. Discussion and Future Work

The Diver tool has been very well received by the developer community. It recently received the honor of being named finalist for the *Best Developer Tool of 2011* by the Eclipse community[2] and it is among the top 150 most-installed Eclipse extensions according to the Eclipse Marketplace[3]. We have also run a user study to help validate Diver on empirical grounds and are preparing the study for publication.

Diver is available as Free and Open Source Software. Further details can be found at: http://diver.sf.net.

V. Acknowledgements

This work is funded by Defence Research and Development Canada under contract W7701-82702/001/QCA, and a DND/NSERC grant with IBM and DRDC Valcartier (DNDPJ 380607-09).

References

[1] K. H. Bennett and V. T. Rajlich, "Software maintenance and evolution: a roadmap," in *Proc. of the Conf. on The Future of Software Engineering*. New York, NY, USA: ACM, 2000, pp. 73–87.

[2] A. Von Mayrhauser and A. Vans, "Program comprehension during software maintenance and evolution," *Computer*, vol. 28, no. 8, pp. 44–55, August 1995.

[3] D. Myers and M.-A. Storey, "Using dynamic analysis to create trace-focused user interfaces for ides (tool demo)," in *Proc. of the 18th ACM SIGSOFT Int'l Symp. on The Foundations of Software Engineering*. ACM, 2010.

[4] M. Kersten and G. C. Murphy, "Using task context to improve programmer productivity," in *Proc. of the 14th ACM SIGSOFT Int'l Symp. on The Foundations of Software Engineering*. New York, NY, USA: ACM, 2006, pp. 1–11.

[5] N. Wilde and M. Scully, "Software reconnaissance: mapping program features to code," *Journal of Software Maintenance: Research and Practice*, vol. 7, no. 1, pp. 49–62, 1995.

[6] J. Starke, *Finding what is important: understanding and improving code search*. University of Calgary, July 2010, masters Thesis.

[7] D. Myers, M.-A. Storey, and M. Salois, "Utilizing debug information to compact loops in large execution traces," in *Proc. of the European Conf. on Software Maintenance and Re-engineering*. IEEE, March 2010, pp. 41–50.

[2]http://www.eclipse.org/org/press-release/20110301_awardfinalists.php

[3]http://marketplace.eclipse.org/metrics/installs