Summary: ICSE Workshop on Dynamic Analysis (WODA 2003)

Jonathan E. Cook
Department of Computer Science
New Mexico State University
Las Cruces, NM 88003 USA
jcook@cs.nmsu.edu

Michael D. Ernst
MIT Lab for Computer Science
Massachusetts Institute of Technology
Cambridge, MA 02139 USA
mernst@lcs.mit.edu

Abstract

Dynamic analysis of software systems has long proven to be a practical approach to gain understanding of the operational behavior of the system. This workshop brought together researchers in the field of dynamic analysis to discuss the breadth of the field, order the field along logical dimensions, expose common issues and approaches, and stimulate synergistic collaborations among the participants.

Introduction

Dynamic analysis encompasses many techniques that reason about the observed dynamic behavior of systems [1]. Examples include assertion checking, memory allocation monitors, and profilers. Dynamic analysis includes both offline techniques that might take substantial compute time and operate on a trace or some other captured representation of the system's behavior, and runtime techniques that operate while the system is producing its behavior. Dynamic analysis techniques have proven useful and practical for many software engineering tasks.

The 2003 ICSE Workshop on Dynamic Analysis (WODA 2003) brought together researchers from around the world, discussing topics from hardware support for instrumentation, to pattern discovery techniques, to using dynamic analysis for test case generation, and many more. The workshop was structured in sessions of short (20-minute) presentations on related topics, followed by questions, interaction, and discussion involving all of the presenters.

In this summary we chose to mention only the presenters by name, for the sake of conciseness. The co-authors are just as important, and many attended the workshop and contributed greatly to its success. We invite readers whose interest is piqued by this summary to access the full program of WODA 2003 at

http://www.cs.nmsu.edu/~jcook/woda2003/

Workshop session summaries

Andreas Zeller began the day with a thought-provoking theory of program analysis that was structured over a hierarchy of reasoning techniques: deduction, observation, induction, and experimentation. Deductive analysis can generate a concrete fact from an abstract program, without executing the program. A debugger or program monitor can observe concrete events from a single execution. Inductive analysis over multiple executions can find a generalization (e.g., an invariant). Finally, controlling the executions through experimentation can allow an analysis to generate specific, strongly substantiated facts about multiple executions.

Instrumentation

A two-paper session on instrumentation followed the opening session. Raimondas Lencevicius began with a look at architecture and efficiency concerns in an industrial setting. His experience in real-time embedded system software led to the conclusion that planned instrumentation must be an explicit part of the development process, because it impacts the performance of the overall system and thus tradeoffs between desired instrumentation and other system concerns must be made.

Markus Mock presented ideas for using low-level mechanisms such as built-in CPU performance counters to understand program-level behavior. Mock discussed how existing hardware mechanisms already built into CPUs can be applied to some well-known program analysis problems, and proposed that dynamic analysis could both benefit from existing hardware mechanisms and inspire new ones that could further aid software development.

Testing and Static Analysis

The session on the relationships of dynamic analysis with testing and static analysis included three papers. Tao Xie presented ideas for improving both the test suites of a program and the inferred specifications from dynamic analysis, such as program invariants. Test cases from automatic test generation tools can be selectively adopted based on their ability to violate existing invariants that were derived from the existing test cases. This avoids bloating the test suite with unneccesary test cases and improves both the testing and the understanding of the program itself.

Neelam Gupta showed how dynamically discovered pro-

gram invariants could be used to automatically generate useful test cases. One goal was to produce test suites that were more useful for dynamic analysis. Gupta showed how discovered invariants could be used to guide test case generation based on path selection.

Finally, Michael Ernst explored the relationship of dynamic analysis and static analysis, and where they might synergistically be applied to be able to perform new analyses or to handle larger programs. Ernst proposed a common framework in which static and dynamic analyses were special cases. He also proposed that researchers from both communities should look to successes on the other side for inspiration and direction on their own side.

Pattern discovery

A three-paper session on pattern discovery followed lunch. Lothar Wendehals presented how dynamic analysis could improve object-oriented design pattern recognition. Dynamic analysis removed a large number of false positives by matching the expected dynamic behavior of patterns found through static structural analysis.

Timothy Lethbridge discussed techniques to reduce extremely large traces of procedure calls to more meaningful information, in particular repeated patterns over the dynamic call tree that the program execution produced. Several separate criteria were presented that together help compress the call tree into patterns that can be related to higher-level domain-oriented concepts rather than low-level implementation details.

Finally, Joel Winstead presented ideas for performing differential program analysis, where the focus is on finding the important behavioral differences between two related programs (e.g., two versions of the same program). While the syntactic difference between versions is easy to detect, finding the inputs that exercise that syntactic difference can be very hard. Winstead presented ideas that use machine learning techniques along with guided fitness criteria to discover the correct inputs that exercise the difference in the versions.

Frameworks and languages

The closing session of the workshop covered frameworks and languages for dynamic analysis. Steve Reiss discussed feature requirements for frameworks and special purpose analysis languages. He proposed to separate the instrumentation from the analysis, even possibly using separate languages for specifying these aspects of a dynamic analysis.

Mikhail Auguston presented general framework issues that arose in building the UFO Dynamic Analysis framework. One example was the disconnect between the desire to write an abstract specification of a specific analysis as if it had a full, post-mortem trace of a program available and the desire to efficiently compute the analysis during runtime, thus avoiding the often prohibitive cost of generating a full trace.

Finally, Jonathan Cook discussed ideas and issues in using existing scripting languages to build program monitoring and dynamic analysis tools. Most scripting languages are relatively easily interfaced with lower-level system programming languages, and this presents an opportunity for easing the cost of implementing ad hoc analyses or prototyping more robust or complete analyses.

Conclusion

During the closing discussion at WODA 2003, the participants decided it was so informative and enjoyable that they wanted to repeat the experience. David Evans and Raimondas Lencevicius have proposed a second workshop on dynamic analysis for ICSE 2004. All interested parties are encouraged to watch for an announcement, and to plan on submitting to and attending both ICSE and WODA 2004!

References

[1] T. Ball. The Concept of Dynamic Analysis. In *Proceedings of the Seventh European Software Engineering Conference / Seventh ACM SIGSOFT Symposium on the Foundations of Software Engineering*, pages 216–234. Springer-Verlag, Sept. 1999.