

David Coppit

Research Statement

Overview

My current research in software engineering tackles two general problems: the difficulties arising from tangling of concerns in code and the difficulties of automated testing. In the past I have performed research in software development methods and applied formal methods, and have made contributions to the field of reliability engineering.

In the area of separation of concerns, much research has focused on the development of programming language abstractions for separating concerns by encapsulating them. Examples include procedures, classes, and more recently aspects and hyperspaces. As successful as these approaches are, they are ineffective at encapsulating concerns that interact with the remainder of the code in idiosyncratic ways. An example of such a concern is debugging code, which is highly context-dependent and nonuniform in its integration with surrounding code.

My research approaches this problem in a radically different way. Rather than seeking to refactor such tightly coupled concerns into new modules, I seek to provide the programmer with tools needed to manage the tangling and interaction of concerns. The approach, which I call *software plans*, provides the programmer with multiple concern-oriented views of a module in much the same way that an architect would work with multiple plans for a building. This work represents a significant departure from the previous language-based approaches, exploring an editor-based approach instead.

In the area of software testing, my research has focused on lowering the cost of testing through automation. Traditional testing approaches such as partition testing can be effective, but scale poorly because of the labor-intensive nature of selecting representative inputs and validating outputs. At the same time, there has been both theoretical and empirical research that demonstrates that partition testing can be less effective than even random testing.

My research seeks to establish the utility of less labor-intensive testing strategies, and to provide tools supporting the generation of structurally-complex test data. My work in *bounded exhaustive testing* has shown that through automation one can test millions of cases, revealing significant faults that were missed by traditional testing strategies. In other work, I have investigated the automatic generation of complex test data such as well-formed web pages, and the automatic generation of implicitly-defined inputs such as those that result in high memory usage.

My previous PhD work sought to improve the state of the art in the development of software supporting more traditional engineering disciplines such as reliability engineering. I developed an approach for developing engineering tools having high usability, functionality, and trustworthiness at low cost. The key was to combine the use of mass-market applications as components for the user interface with the judicious use of formal methods for the analysis core. In working closely with reliability engineers, I also found opportunities to apply concepts from software engineering and programming languages to problems in their field, making some modest contributions.

The next two sections describe my current research in software plans and bounded exhaustive testing. The next section describes my previous work in software development methods. I conclude with my future research agenda.

Software Plans

For thirty years computer scientists have struggled with the complexity of software, developing new abstractions for achieving separation of concerns through encapsulation. This approach is most successful when concerns interact in narrow, well-defined ways. Unfortunately, many concerns (such as debugging tracing) exhibit complex tangling that is difficult to specify in a manner amenable to automated integration by a compiler. As a result, such concerns are usually left tangled in the code, increasing the difficulties of software development and maintenance.

My research tackles this problem in a novel way—instead of separating concerns by refactoring them into new modules, I treat a module as a *multidimensional* representation of the concerns that it implements. That is, instead of a single flat code representation, my work in an area I call *software plans* represents a module as a set of inter-related, concern-oriented views. A software plan is an editable view of a module that presents the relevant subset of the code for the concern being implemented.

This work has the potential to create a new direction for research in separation of concerns in code. My recent work has focused on laying the foundation for more significant advances in this area. For example, we developed a new code representation for software plans, implemented a prototype editor called Spotlight, and developed techniques for reverse-engineering software plans from existing code.

1. David Coppit, Robert Painter, and Meghan Revelle. Spotlight: A Prototype Tool for Software Plans. In *Proceedings of 2007 the International Conference on Software Engineering*, pages 754-757, Minneapolis, Minnesota, 20-26 May 2007. IEEE. Refereed formal research demonstration paper.
2. Robert R. Painter and David Coppit. A Model for Software Plans. In *Proceedings of the First International Workshop on the Modeling and Analysis of Concerns in Software*, pages 14-18, St. Louis, Missouri, 16 May 2005.
3. Meghan Revelle, Tiffany Broadbent, and David Coppit. Understanding Concerns in Software: Insights Gained from Two Case Studies. In *Proceedings of 13th IEEE International Workshop on Program Comprehension*, pages 23-32, St. Louis, Missouri, 15-16 May 2005.
4. David Coppit and Benjamin Cox. Software plans for separation of concerns. In *Proceedings of the Third AOSD Workshop on Aspects, Components, and Patterns for Infrastructure Software*, Lancaster, UK, 22 March 2004. IEEE.

This work is supported by a \$328K grant from the United States Air Force Office of Scientific Research. We are starting the second of three years of support.

Bounded Exhaustive Testing

Traditional testing involves partitioning the input space, selecting a representative, and then deriving the expected output or verifying the actual output. Selection of cases to test is driven by models of the correlation between faults and the design or implementation of the software. As effective as this approach can be, it suffers from a key problem: there is little empirical proof that the fault model is sufficient to reveal all important faults. That is, techniques such as code coverage may reveal unimportant faults while also missing important faults. Both empirical and theoretical research have shown, for example, that partition-based approaches can be worse than random testing if the partitioning is poor.

My research has investigated alternative automated testing strategies such as *bounded exhaustive testing*, where a subset of the program's input space is identified and all inputs within that space are tested. This approach is attractive because it places few assumptions on the distribution of fault-revealing inputs. In a study I conducted with colleagues at MIT and the University of Virginia, we showed the utility of the approach, revealing faults in a dynamic fault tree evaluation engine by testing millions of trees. We also developed techniques for scaling up the generation of inputs using model checking technologies.

This work also led me to further investigate the problems of automated input generation. In one effort, I developed a tool that takes as input a lex/yacc-style specification and generates a generator for strings in the language. Assuming a programmer uses lex and yacc to create a parser for the input, my tool allows them to use the input files to create a test data generator instead. In other work, a student and I developed an algorithm for automatically identifying the bounds of an "interesting" subset of the input space. We evaluated the algorithm by using it to generate and test near miss trajectories for an aircraft collision-avoidance algorithm.

1. John A. Murphy and David Coppit. Random Generation of Test Inputs for Implicitly Defined Subdomains. In *Proceedings, ICSE 2007 Workshops: Second International Workshop on Automation of Software Test*, pages 13-16, Minneapolis, Minnesota, 26 May 2007.
2. David Coppit and Jixin Lian. yacc: An Easy-To-Use Generator for Structured Test Inputs. In *20th IEEE/ACM International Conference on Automated Software Engineering*, pages 356-359, Long Beach, California, 7-11 November 2005.
3. David Coppit, Jinlin Yang, Sarfraz Khurshid, Wei Le, and Kevin Sullivan. Software assurance by bounded exhaustive testing. *IEEE Transactions on Software Engineering*, 31(4):328-39, April 2005.
4. David Coppit and Jennifer Haddox-Schatz. On the use of specification-based assertions as test oracles. In *Proceedings of the 29th Annual IEEE/NASA Software Engineering Workshop*, Greenbelt, Maryland, 6-7 April 2005.
5. Kevin Sullivan, Jinlin Yang, David Coppit, Sarfraz Khurshid, and Daniel Jackson. Software assurance by bounded exhaustive testing. In *Proceedings of the International Symposium on Software Testing and Analysis*, Boston, Massachusetts, 11-14 July 2004.

This work was supported by a \$55K subcontract with University of Virginia on a NASA grant. Unfortunately only \$37K was disbursed due to NASA budget shortfalls.

Previous Research

If engineering is the application of science to the development of artifacts for the good of mankind, I believe that software engineering's best hope for leveraging science is to employ *formal methods*—mathematically-based tools, techniques and processes. As a graduate student, my research focused on finding cost-effective ways of employing formal methods for the construction of more reliable software. My primary application area was that of reliability engineering software, where engineers use domain-specific languages to create models and analyze them with respect to properties such as reliability over time.

In my interactions with engineers, I found that the languages they were developing were pushing the boundaries of their ability to specify their correct behavior. Further, the tools they developed lacked a level of usability and quality that users expect. My research addressed these challenges by combining component-based software development techniques and applied formal methods. Formal methods aid in the design and validation of the modeling language, and in the implementation of a trustworthy analysis library. *Package-oriented programming*, the use of mass-market applications as components, provides the features and usability of tools at low cost. To evaluate this approach, I applied it to an important modeling method, dynamic fault tree analysis. My work was a significant contribution to the *Galileo* tool used by space station engineers at NASA and now sold by Exelix, Inc.

1. David Coppit and Kevin J. Sullivan. Sound methods and effective tools for engineering modeling and analysis. In *Proceedings of the 25th International Conference on Software Engineering*, pages 198-207, Portland, Oregon, 3-10 May 2003. IEEE.
2. David Coppit and Kevin J. Sullivan. Multiple mass-market applications as components. In *Proceedings of the 22nd International Conference on Software Engineering*, pages 273-82, Limerick, Ireland, 4-11 June 2000. IEEE.
3. David Coppit and Kevin J. Sullivan. Galileo: A tool built from mass-market applications. In *Proceedings of the 22nd International Conference on Software Engineering*, pages 750-3, Limerick, Ireland, 4-11 June 2000. IEEE.
4. Joanne Bechta Dugan, Kevin J. Sullivan, and David Coppit. Developing a high-quality software tool for fault tree analysis. In *Proceedings of the International Symposium on Software Reliability Engineering*, pages 222-31, Boca Raton, Florida, 1-4 November 1999. IEEE.
5. Kevin J. Sullivan, David Coppit, and Joanne Bechta Dugan. The Galileo fault tree analysis tool. In *Proceedings of the 29th Annual International Symposium on Fault-Tolerant Computing*, pages 232-5, Madison, Wisconsin, 15-18 June 1999. IEEE.
6. David Coppit and Kevin J. Sullivan. Formal specification in collaborative design of critical software tools. In *Proceedings Third IEEE International High-Assurance Systems Engineering Symposium*, pages 13-20, Washington, D.C., 13-14 November 1998. IEEE.
7. Kevin J. Sullivan, Jake Cockrell, Shengtong Zhang and David Coppit, Package-oriented programming of engineering tools. In *Proceedings of 1997 International Conference on Software Engineering*, pages 616-7, Boston, Massachusetts, 17-23 May 1997. IEEE. Refereed formal research demonstration paper.

In working closely with reliability engineers, I have also been able to make some modest contributions to the field of reliability engineering. This work has primarily focused on the utility of using formal specification languages to provide a denotational semantics for reliability modeling languages. More recent work has focused on demonstrating the utility of an intermediate reliability modeling language.

1. David Coppit, Robert R. Painter. High-level reliability languages using a general intermediate domain. In *Proceedings of the 51st Annual Reliability and Maintainability Symposium*, Alexandria, Virginia, 24-27 January 2005. IEEE.
2. David Coppit, Robert R. Painter, and Kevin J. Sullivan. Shared semantic domains for computational reliability engineering. In *Proceedings of the International Symposium on Software Reliability Engineering*, pages 168-180, Denver, Colorado, 17-20 November 2003. IEEE
3. Joanne Bechta Dugan, Kevin J. Sullivan, and David Coppit. Developing a low-cost high-quality software tool for dynamic fault tree analysis. *Transactions on Reliability*, March 2000, pages 49-59. IEEE.
4. David Coppit, Kevin J. Sullivan, and Joanne Bechta Dugan. Formal semantics of models for computational engineering: a case study on dynamic fault trees. In *Proceedings of the International Symposium on Software Reliability Engineering*, pages 270-282, San Jose, California, 8-11 October 2000. IEEE.
5. Ragavan Manian, David Coppit, Kevin Sullivan, and Joanne Bechta Dugan. Bridging the gap between systems and dynamic fault tree models. In *Annual Reliability and Maintainability Symposium 1999 Proceedings*, pages 105-11, Washington, D.C., 18-21 January 1999. IEEE.
6. Ragavan Manian, Joanne Bechta Dugan, David Coppit, and Kevin Sullivan. Combining various solution techniques for dynamic fault tree analysis of computer systems. In *Proceedings Third IEEE International High-Assurance Systems Engineering Symposium*, pages 21-28, Washington, D.C., 13-14 November 1998. IEEE.

Future Research

My future work will focus primarily on software plans and testing. Work on software plans is progressing on several fronts. We have recently developed a dynamic approach for reverse-engineering software plans from legacy code and plan to improve the approach by adding static analysis. The software plans approach makes manifest feature interactions that occur in software. We hope to develop analyses for automatically detecting some classes of feature interaction and helping the programmer to resolve them. In other work, we are augmenting our existing lexical code representation with program semantics in order to ease the integration of separately developed software plans. All of these activities and more are supported by an Air Force grant.

With regard to testing, short-term plans include implementing recent optimizations developed in the model checking in my input generation tool. These can potentially improve the generation efficiency by several orders of magnitude. In longer-term work, I plan to create a database of programs, operational profiles, and fault-revealing inputs. Much research in software testing is hampered by our inability to rapidly and quantitatively evaluate testing strategies. By creating such a repository for the public, researchers will have a common benchmark for comparing testing strategies, including their effect on overall program reliability.

Nearly all of my work to date has involved graduate and undergraduate collaborators. I plan to continue to involve students, as well as to continue to establish collaborations with people outside, and hopefully within my department.