Spring 2018
EE382C-3 Verification and Validation of Software
(Unique: 16049)

Instructor

Sarfraz Khurshid
EER 7.880
(512) 471-8244
khurshid@ece.utexas.edu
Office hours: by appointment

Teaching Assistants

Kaiyuan Wang
kaiyuanw@utexas.edu
Office hours: by appointment

Mengshi Zhang
mengshi0617@gmail.com
Office hours: by appointment

Time and Location

All lectures will be 1:00-5:00pm at ECJ 1.316.

Prerequisites

The students are expected to have basic knowledge of discrete math (e.g., sets, relations and propositional logic), data structures (e.g., linked-lists and binary trees) and object-oriented programming (preferably in Java, C++, or C#) and considerable programming experience.

Catalog entry

Advanced concepts and techniques for checking the correctness of programs and specifications. Topics include state-of-the-art research in systematic software testing, state-space exploration techniques, including symbolic techniques, for software model checking, heuristics-based approaches, as well as static program analyses, including those based on automated theorem proving, and behavioral specification languages.

Description

The process of software validation includes reasoning about (the correctness of) programs, whether formally—a process that is termed verification—or informally, and testing programs. This course focuses on verification and testing. A NIST report from 2002 estimates that software failures cost the US economy $59.5 billion dollars annually and over a third of this cost could be saved using a better infrastructure for testing. It is widely accepted that testing currently accounts for more than one half of the cost of software development. Learning the techniques and tools presented in this course is likely to significantly increase the students' productivity as software developers and testers and improve the quality of the code they develop.

The course is organized as a series of lectures on basics of software testing as well as more advanced research/tool papers. The selected papers will cover traditional and state-of-the-art techniques for software testing and verification. (An initial list of candidate papers will be provided at the beginning of class. Different papers may be selected in view of class preferences.) The course content will cover both techniques for dynamic analysis, such as glass box and black box testing, equivalence partitioning, boundary value analysis, test strategy and automation, regression testing and debugging, techniques for static analysis, and also techniques for software model checking including those that employ heuristics based on artificial intelligence.

Grading

The grade will be based on class participation (10%), homeworks (20%), exams (40%), and a final group project (30%). Students must participate actively in the class. The final project will be done in a group of two or three students. A typical project would involve performing a case study using some tool(s) studied in the class. With instructor's permission, the students may choose to work on a suitable idea of their own. Good projects will result in work that is of a quality expected for conference/workshop publication. At the end of the course, students will present their projects to the class.

Textbooks (recommended)

The following texts provide some basic material that would help students understand the more advanced material in the papers:

1. *Introduction to Software Testing* by Paul Amman and Jeff Offutt. ISBN: 9781107172012
2. *Software Abstractions: Logic, Language, and Analysis* by Daniel Jackson. Revised edition, ISBN: 0262017156

Calendar (tentative)

Note the following tentative schedule for the problem sets and exams:

```
Problem Set 1
    OUT: January 20; DUE: February 12

Problem Set 2 [Project Proposal]
    OUT: January 20; DUE: February 28

Problem Set 3
    OUT: February 10; DUE: March 5

Problem Set 4 [Project Update]
    OUT: March 1; DUE: March 31

Problem Set 5 [Final Project Report]
    OUT: April 1; DUE: April 30


Exam 1: March 3

Exam 2: May 5
```

Students with disabilities

Students with disabilities may request appropriate academic accommodations from the Division of Diversity and Community Engagement, Services for Students with Disabilities (Tel: 512-471-6259; online: http://www.utexas.edu/diversity/ddce/ssd/).

ECE's academic honesty statement

Faculty in the ECE Department are committed to detecting and responding to all instances of scholastic dishonesty and will pursue cases of scholastic dishonesty in accordance with

university policy. Scholastic dishonesty, in all its forms, is a blight on our entire academic community. All parties in our community—faculty, staff, and students—are responsible for creating an environment that educates outstanding engineers, and this goal entails excellence in technical skills, self-giving citizenry, an ethical integrity. Industry wants engineers who are competent and fully trustworthy, and both qualities must be developed day by day throughout an entire lifetime. Scholastic dishonesty includes, but is not limited to, cheating, plagiarism, collusion, falsifying academic records, or any act designed to give an unfair academic advantage to the student. The fact that you are in this class as an engineering student is testament to your abilities. Penalties for scholastic dishonesty are severe and can include, but are not limited to, a written reprimand, a zero on the assignment/exam, re-taking the exam in question, an F in the course, or expulsion from the University. Don't jeopardize your career by an act of scholastic dishonesty. Details about academic integrity and what constitutes scholastic dishonesty can be found at the website for the UT Dean of Students Office and the General Information Catalog, Section 11-802.

References

[1] M. Barnett, R. DeLine, M. Fahndrich, K. R. M. Leino, and W. Schulte. Verification of object-oriented programs with invariants. In *Proc. ECOOP 2003 Workshop on Formal Techniques for Java-like Programs (FTfJP)*, 2003.

[2] M. Barnett, W. Grieskamp, L. Nachmanson, W. Schulte, N. Tillmann, and M. Veanes. Model-based testing with asml.net. In *1st European Conference on Model-Driven Software Engineering*, December 2003.

[3] M. Barnett, K. R. M. Leino, and W. Schulte. The Spec# programming system: An overview. In *Proc. Cassis International Workshop*, 2004.

[4] M. Barnett and W. Schulte. Runtime verification of .NET contracts. *Journal of Systems and Software*, 65(3), 2003.

[5] D. Beyer, A. J. Chlipala, T. A. Henzinger, R. Jhala, and R. Majumdar. Generating tests from counterexamples. In *Proc. 26th International Conference on Software Engineering (ICSE)*, pages 326–335, 2004.

[6] C. Boyapati, S. Khurshid, and D. Marinov. Korat: Automated testing based on Java predicates. In *Proc. International Symposium on Software Testing and Analysis (ISSTA)*, pages 123–133, July 2002.

[7] W. R. Bush, J. D. Pincus, and D. J. Sielaff. A static analyzer for finding dynamic programming errors. *Software—Practice and Experience*, 30(7):775–802, 2000.

[8] D. M. Cohen, S. R. Dalal, M. L. Fredman, and G. C. Patton. The AETG system: An approach to testing based on combinatorial design. *IEEE Transactions on Software Engineering*, 23(7):437–444, 1997.

[9] C. Cowan, P. Wagle, C. Pu, S. Beattie, and J. Walpole. Buffer overflows: Attacks and defenses for the vulnerability of the decade. In *Proc. DARPA Information Survivability Conference and Exposition (DISCEX)*, Jan. 2000.

[10] S. R. Dalal, A. Jain, N. Karunanithi, J. M. Leaton, C. M. Lott, G. C. Patton, and B. M. Horowitz. Model-based testing in practice. In *ICSE '99: Proceedings of the 21st international conference on Software engineering*, pages 285–294, Los Alamitos, CA, 1999.

[11] B. Demsky and M. Rinard. Automatic detection and repair of errors in data structures. In *Proc. ACM SIGPLAN 2003 Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)*, pages 78–95, 2003.

[12] M. D. Ernst, J. Cockrell, W. G. Griswold, and D. Notkin. Dynamically discovering likely program invariants to support program evolution. *IEEE Transactions on Software Engineering*, 27(2):99–123, 2001.

[13] C. Flanagan, K. R. M. Leino, M. Lillibridge, G. Nelson, J. B. Saxe, and R. Stata. Extended static checking for Java. In *Proc. ACM SIGPLAN 2002 Conference on Programming language design and implementation*, pages 234–245, 2002.

[14] S. M. Freudenberger, J. T. Schwartz, and M. Sharir. Experience with the SETL optimizer. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 5(1), 1983.

[15] P. Godefroid. Model checking for programming languages using VeriSoft. In *Proc. 24th Annual ACM Symposium on the Principles of Programming Languages (POPL)*, pages 174–186, Paris, France, Jan. 1997.

[16] P. Godefroid and S. Khurshid. Exploring very large state spaces using genetic algorithms. In *Proc. 8th International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS)*, Grenoble, France, Apr. 2002.

[17] P. Godefroid, N. Klarlund, and K. Sen. DART: Directed automated random testing. In *Proc. 2005 ACM SIGPLAN conference on Programming language design and implementation (PLDI)*, 2005.

[18] A. Groce and W. Visser. Heuristics for model checking java programs. *International Journal on Software Tools for Technology Transfer (STTT)*, 6(4):260–276, 2004.

[19] S. Hangal and M. S. Lam. Tracking down software bugs using automatic anomaly detection. In *Proc. 24th International Conference on Software Engineering (ICSE)*, May 2002.

[20] M. Harder, J. Mellen, and M. D. Ernst. Improving test suites via operational abstraction. In *Proc. 25th International Conference on Software Engineering*, pages 60–71, Portland, OR, May 2003.

[21] J. Henkel and A. Diwan. Discovering algebraic specifications from java classes. In *Proc. European Conference on Object-Oriented Programming (ECOOP)*, July 2003.

[22] J. Henkel and A. Diwan. A tool for writing and debugging algebraic specifications. In *Proc. 26th International Conference on Software Engineering (ICSE)*, 2004.

[23] D. Jackson. Micromodels of software: Modelling and analysis with Alloy, 2001. `http://sdg.lcs.mit.edu/alloy/book.pdf`.

[24] D. Jackson and M. Vaziri. Finding bugs with a constraint solver. In *Proc. International Symposium on Software Testing and Analysis (ISSTA)*, Portland, OR, Aug. 2000.

[25] S. Khurshid. Testing an intentional naming system using genetic algorithms. In *Proc. 7th International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS)*, Genova, Italy, Apr. 2001.

[26] S. Khurshid, I. García, and Y. L. Suen. Repairing structurally complex data. In *Proc. 12th SPIN Workshop on Software Model Checking*, 2005.

[27] S. Khurshid and D. Marinov. TestEra: Specification-based testing of Java programs using SAT. *Automated Software Engineering Journal*, 2004.

[28] S. Khurshid, D. Marinov, and D. Jackson. An analyzable annotation language. In *Proc. ACM SIGPLAN 2002 Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)*, Seattle, WA, Nov 2002.

[29] S. Khurshid, C. Pasareanu, and W. Visser. Generalized symbolic execution for model checking and testing. In *Proc. 9th International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS)*, Warsaw, Poland, April 2003.

[30] J. C. King. Symbolic execution and program testing. *Communications of the ACM*, 19(7):385–394, 1976.

[31] J. R. Larus, T. Ball, M. Das, R. DeLine, M. Fhndrich, J. Pincus, S. K. Rajamani, and R. Venkatapathy. Righting software. *IEEE Software*, 21(3):92–100, 2004.

[32] G. T. Leavens, A. L. Baker, and C. Ruby. Preliminary design of JML: A behavioral interface specification language for Java. Technical Report TR 98-06i, Department of Computer Science, Iowa State University, June 1998. (last revision: Aug 2001).

[33] X. Li, D. Shannon, J. Walker, S. Khurshid, and D. Marinov. Analyzing the uses of a software modeling tool. In *Proc. Sixth Workshop on Language Descriptions, Tools and Applications (LDTA)*, Vienna, Austria, Apr. 2006.

[34] D. Marinov and S. Khurshid. VAlloy: Virtual functions meet a relational language. In *Proc. Formal Methods Europe (FME)*, Copenhagen, Denmark, July 2002.

[35] D. Marinov, S. Khurshid, S. Bugrara, L. Zhang, and M. Rinard. Optimizations for compiling declarative models into boolean formulas. In *8th International Conference on Theory and Applications of Satisfiability Testing (SAT 2005)*, St.Andrews, Scotland, 2005.

[36] T. Millstein. Practical predicate dispatch. In *Proc. ACM SIGPLAN 2004 Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)*, Oct. 2004.

[37] J. S. Moore and G. Porter. The apprentice challenge. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 24(3), 2002.

[38] M. Musuvathi and D. Engler. Model checking large network protocol implementations. In *Proc. First Symposium on Networked Systems Design and Implementation*, pages 155–168, 2004.

[39] Robby, E. Rodríguez, M. Dwyer, and J. Hatcliff. Checking strong specifications using an extensible software model checking framework. In *Proc. 10th International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS)*, 2004.

[40] D. Saff and M. Ernst. An experimental evaluation of continuous testing during development. In *Proc. Proc. International Symposium on Software Testing and Analysis (ISSTA)*, 2004.

[41] S. Savage, M. Burrows, G. Nelson, P. Sobalvarro, and T. Anderson. Eraser: A dynamic data race detector for multithreaded programs. *ACM Transactions on Computer Systems*, 15(4):391–411, 1997.

[42] E. G. Sirer and B. N. Bershad. Using production grammars in software testing. In *Proc. 2nd conference on Domain-specific languages*, pages 1–13, 1999.

[43] W. Visser, K. Havelund, G. Brat, and S. Park. Model checking programs. In *Proc. 15th IEEE International Conference on Automated Software Engineering (ASE)*, Grenoble, France, 2000.

[44] C. von Praun and T. R. Gross. Static conflict analysis for multi-threaded object-oriented programs. In *Proc. ACM SIGPLAN'03 Conference on Programming Language Design and Implementation (PLDI)*, June 2003.

[45] J. Whaley, M. C. Martin, and M. S. Lam. Automatic extraction of object-oriented component interfaces. In *Proc. International Symposium on Software Testing and Analysis (ISSTA)*, July 2002.

[46] T. Xie, D. Marinov, W. Schulte, and D. Notkin. Symstra: A framework for generating object-oriented unit tests using symbolic execution. In *Proc. 11th International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS)*, Edinburgh, UK, Apr. 2005.

[47] F. Zaraket, A. Aziz, and S. Khurshid. Sequential encoding for relational analysis. In *Proc. 29th International Conference on Software Engineering (ICSE)*, 2007.

[48] A. Zeller. Yesterday, my program worked. today, it does not. why? In *Proc. 7th ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE)*, pages 253–267, 1999.

[49] A. Zeller. Isolating cause-effect chains from computer programs. In *Proc. 10th ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE)*, pages 1–10, 2002.