Test Suite Reduction and Prioritization with Call Trees

A Tool Paper Presented at the 22nd IEEE/ACM International Conference on Automated Software Engineering

Atlanta, Georgia, November 5-9, 2007

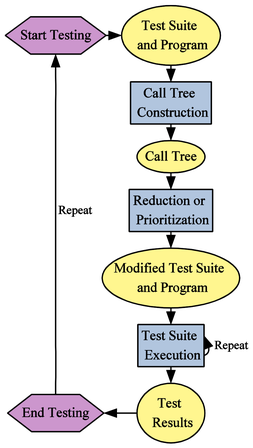
Adam M. Smith
adammatthewsmith@gmail.com

Joshua J. Geiger
joshua.geiger@gmail.com

Gregory M. Kapfhammer
gkapfham@allegheny.edu

Mary Lou Soffa
soffa@cs.virginia.edu

Regression Testing Challenges



- Execution time of a test suite may be prohibitive.
- **Coverage monitoring** supports regression testing by tracking the coverage of test requirements.
- **Call trees** efficiently store the dynamic behavior of a program under test.
- The coverage of the tests can be examined to see where there is **coverage overlap**.
- **Reduction** attempts to eliminate test overlap.
- **Prioritization** aims to rapidly cover the test requirements.

Figure 1: Reduction and prioritization tool.

Test Coverage Monitoring

Probe insertion:

- Insert probes using AspectJ.
- **Static instrumentation** – Probes are inserted directly into the Java bytecode.
- **Dynamic instrumentation** – Probes are inserted when the Java class is loaded.

Probe location:

- Before and after each test case in the test suite.
- Before and after each method in the program.

Implementation:

- Dynamic probes are inserted using class loaders or the JVMTI.
- Call trees are stored in binary or XML.

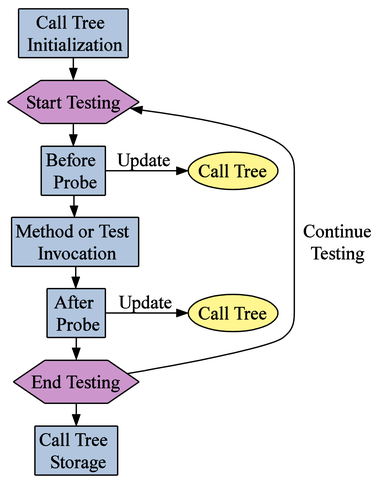


Figure 2: Probes and Call Tree Construction.

Call Trees

Call Tree: tree-based representation of the dynamic behavior of the program during testing.

Call Tree Path: Each path from the root node to a leaf node is considered a requirement [McMaster and Memon, 2005].

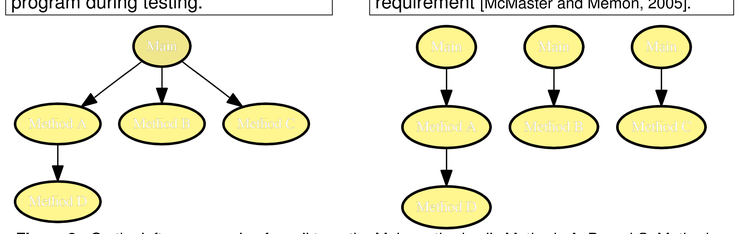


Figure 3: On the left, an example of a call tree; the Main method calls Methods A, B, and C. Method A calls Method D. On the right, the individual call tree paths of the tree on the left.

Reduction Techniques

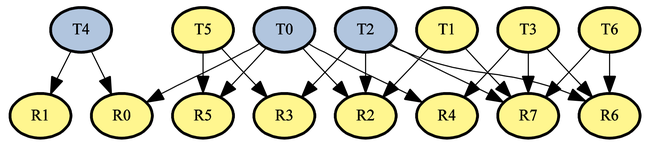


Figure 4: Reduction techniques eliminate overlapping coverage in the test suite. Tests T2, T9 , and T6 cover every requirement; the remaining tests are redundant.

Traditional Greedy Algorithm

- Greedily choose tests by coverage, cost, or the ratio of coverage to cost.

2-Optimal Algorithm [Harman et al, 2007]

- All-pairs comparison of the tests' coverage.
- Generalizes to K-Way.

Harrold, Gupta, Soffa Algorithm [Harrold et al, 1993]

- The tests that cover sparsely covered requirements are more likely to be needed in the final test suite.
- Greedily choose tests by coverage from covering sets of increasing cardinality.

Delayed Greedy Algorithm [Tallam and Gupta, 2005]

- Remove tests whose set of covered requirements is a subset of another test's set of covered requirements.
- Remove requirements whose set of covering tests is a superset of another requirement's set of covering tests.
- Add tests to the test suite that are the only covering test for a requirement, or greedily choose a test based on coverage.

Prioritization Methods

- Each reduction algorithm produces a subset of the original test suite.
- The reduction algorithm can be repeated on the set of tests that did not get chosen. The output from the repeated execution of the algorithm can be added to the output from the previous run.
- By running the reduction algorithm on the residual tests until every test has been chosen, a reordering of the test will be obtained that covers all of the requirements faster than the original test suite.

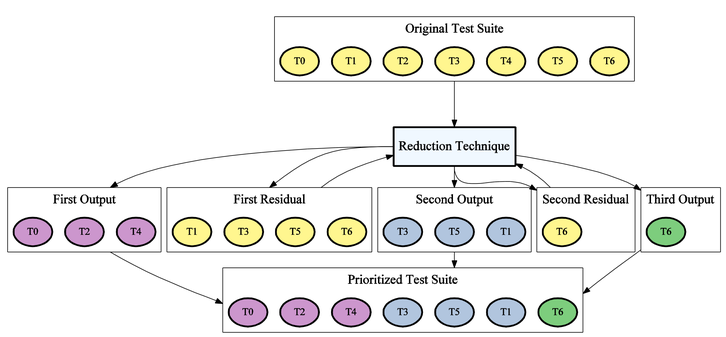


Figure 5: Repeated reduction on residual tests prioritizes the test suite.

Experimental Analysis

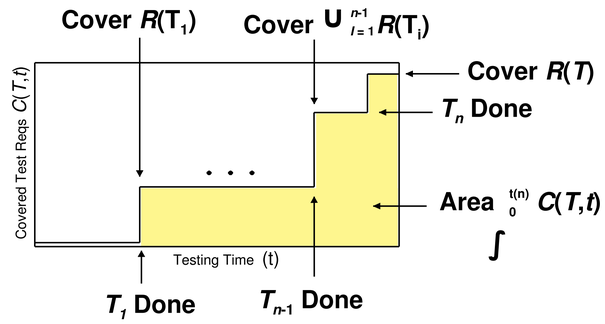


Figure 6: Cumulative coverage.

- Prioritization is rated by the test suite's **Coverage Effectiveness:** the ratio between the area of the cumulative coverage of a prioritized test suite and an optimal test suite.
- Prioritizing to maximize coverage effectiveness ensures requirements are covered earlier during the testing process.

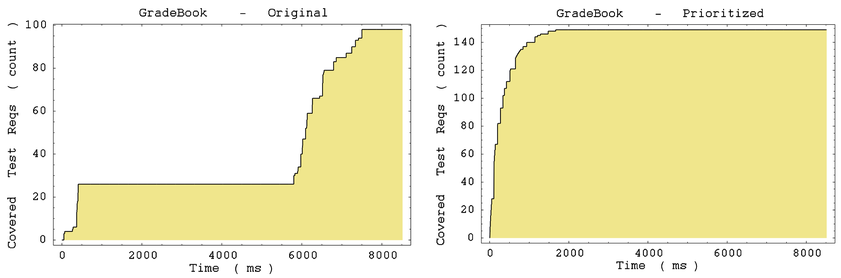


Figure 7: Cumulative coverage of the GradeBook application. On the left, the original test suite has a poor coverage effectiveness. The prioritized test suite on the right covers the requirements more rapidly.

Future Contributions

- **Software Testing Tools:**
  - We are releasing the source code, tutorials, and stand-alone tools in phases.
  - A coverage effectiveness calculator is already available for download.
- **Future Research:**
  - Empirically evaluate the efficiency and effectiveness of the regression testing algorithms using both real world and synthetic test suites.
  - Compare the existing algorithms to search-based methods including hill climbing, genetic algorithms, and simulated annealing.

References

M. Jean Harrold, Rajiv Gupta, and Mary Lou Soffa. A Methodology for Controlling the Size of a Test Suite. *ACM Transactions on Software Engineering Methodologies* 2(3): pages 270-285, 1993.

Sriraman Tallam and Neelam Gupta. A Concept Analysis Inspired Greedy Algorithm for Test Suite Minimization. *Program Analysis for Software Tools and Engineering*, pages 1-8, 2005.

Mark Harman, Zheng Li, and Robert M. Hierons. Search Algorithms for Regression Test Case Prioritization. *IEEE Transactions on Software Engineering*, pages 1-12, 2007.

Scott McMaster and Atif Memon. Call Stack Coverage for Test Suite Reduction. *IEEE International Conference on Software Maintenance,* pages 539-548, 2005.

http://cs.allegheny.edu/~gkapfham/research/kanonizo/