Interactive Coverage Effectiveness Multiplots for Evaluating Prioritized Regression Test Suites

Adam M. Smith†, Joshua J. Geiger†, Gregory M. Kapfhammer‡, Manos Renieris°, and G. Elisabeta Marai†

†Department of Computer Science, University of Pittsburgh ‡Department of Computer Science, Allegheny College °Google

Presented at the IEEE Information Visualization Conference (IEEE InfoVis 2009), Atlantic City, NJ

Software Testing Challenges

- **Complex** source code, databases, files, and network communication
- Defects may exist in the individual **components** or their **interactions**
- Testing **isolates defects** and establishes **confidence** in the correctness of software

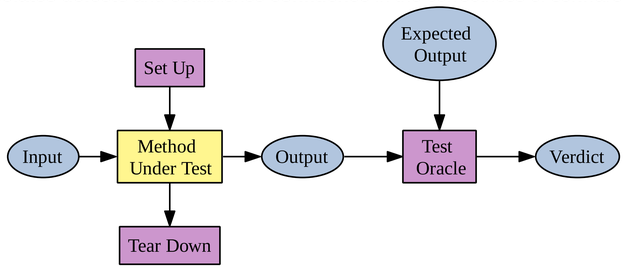


Figure 1: **What is a test case?** Each test case invokes a method within the program and compares the actual and expected output values.

- A sequence of test cases is a **test suite**
- A test suite **executor** such as JUnit runs each test case **independently**

Regression Testing

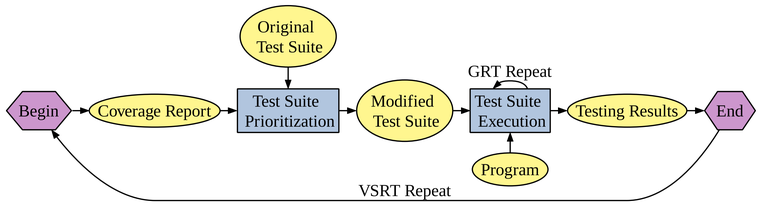
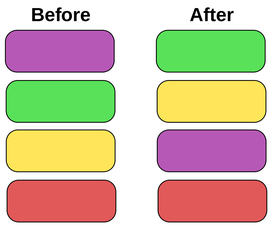


Figure 2: **Regression testing**. A test suite will be executed repeatedly throughout development, searching for faults introduced by changes made to the software.



- When software is **modified**, new tests run in addition to the old, increasing the test suite size
- Execution time of a test suite may be **prohibitive**
- **Prioritization** techniques **re-order** the tests to locate defects earlier in the test execution process [6, 7]
- **Coverage reports** identify points in the source code executed, or **covered**, by each test case
- Prioritizers must analyze the **requirements** covered by each test case to effectively re-order the test suite

Evaluating Test Suites

- **Coverage Effectiveness** (CE) rates test suites by how quickly they cover each requirement [4]
- Prioritize to **increase** the CE of a test suite where $CE = \frac{\text{Actual}}{\text{Ideal}} \in [0, 1]$

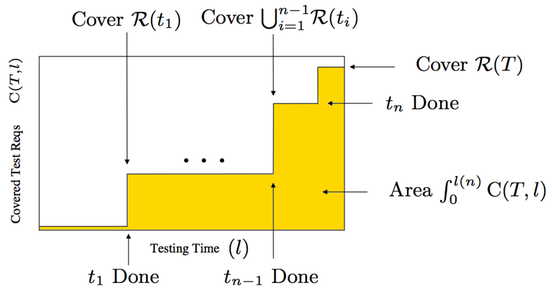


Figure 3: **Calculating Coverage Effectiveness (CE).** The CE score is the area under $C(T, l)$ divided by the area under the ideal test suite function (dashed line). Cover $\mathcal{R}(t_i)$ denotes the set of requirements covered by test t_i

Limitations for Testers

- Many **prioritization** methods exist because finding the **highest** CE by evaluating **all** orderings of a test suite is too **expensive**
- **Each** of these prioritization techniques can have **many configurations** from which to choose
- Testers relying on **static** coverage effectiveness multiplots, such as Figure 4, and/or **large tables** of CE scores and test orders can be easily **overwhelmed**
- **Existing** visualizations assist during **different** development processes such as manual debugging and automated fault localization [2, 3, 5]

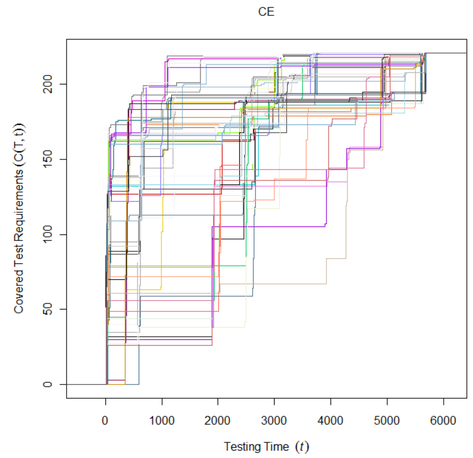


Figure 4: **Static Coverage Effectiveness Multiplot.** Multiple lines severely clutter the visualization making evaluation and comparison of prioritized test suites nearly impossible.

Visualization Design Goals

- Enable software testers to **quickly** find the **best** test suite order for their own applications
- **Interactively** pick prioritizers, comparing CE values and the actual ordering of the tests
- Utilize prioritization techniques such as **greedy** (GRD), **2-optimal greedy** (2OPT), **delayed greedy** (DGR), and **Harrold Gupta Soffa** (HGS) which use greedy choice metrics (GCMs) to efficiently construct new test orders [7]

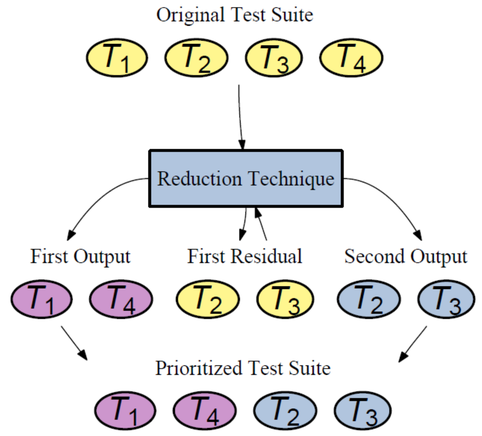


Figure 5: **Greedy approaches to test prioritization.** Re-order the test suite by repeatedly performing reduction.

- Make use of the potential power of **reverse** and **random** prioritizations [7]
- **Visualization** and **UI features** demonstrated by Becker et al. [1] and a NY Times interactive visualization of market statistics (`http://www.nytimes.com/interactive/2008/10/11/business/20081011BEARMARKETS.html`)
- Encourage **empirical study** on the use of **visualization** during test suite prioritization

Visualization Features

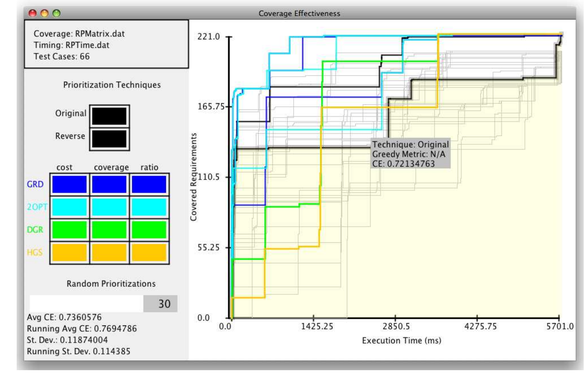


Figure 6: **Interactive Coverage Effectiveness Multiplot in RAISE.** Visualization using Interactive Multiplots and details on demand allows the users to quickly filter, evaluate and compare prioritized test suites.

Left Panel

- Displays information about the test suite and controls multiplot display
- Toggle display of cumulative coverage functions for each prioritization method
- Color coding of prioritization techniques for easy identification
- Adjust the number of random prioritizations
- Display cumulative averages and standard deviations of random prioritizations

Right Panel

- Multiplot of cumulative coverage step functions of all selected prioritization methods
- Mouse-over of plots highlights the line and shades the area under the line
- Mouse-over a line reveals its corresponding prioritizer, GCM, and CE score
- Vertical axis to display the number of covered requirements and horizontal axis to show the test suite execution time

Conclusions and Future Work

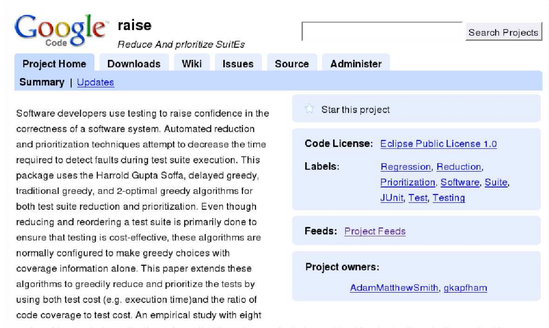


Figure 7: `http://raise.googlecode.com/` provides tools, data sets and resources.

- An **interactive** visualization that enables the **evaluation** of prioritized regression test suites
- Free and open source **Reduce And prIoritize SuitEs** (RAISE) system available for download
- Intend to add new **features** and conduct more experimental **studies**
- Will **extend** RAISE to support other **metrics** like average percentage of faults detected (APFD) and average percentage of requirements covered (APRC)
- RAISE will serve as a **simple** and **valuable** tool in a comprehensive framework supporting all of the phases in the **regression testing** process

References

[1] R. A. Becker, S. G. Eick, and A. R. Wilks. Visualizing Network Data. *IEEE Trans. on Visual. and Comput. Graph.*, 1:16–28, 1995.

[2] J. A. Cottam, J. Hursey, and A. Lumsdaine. Representing unit test data for large scale software development. In *Proc. of 4th SoftVis*, 2008.

[3] J. A. Jones, M. J. Harrold, and J. Stasko. Visualization of test information to assist fault localization. In *Proc. of 24th ICSE*, 2002.

[4] G. M. Kapfhammer and M. L. Soffa. Using coverage effectiveness to evaluate test suite prioritizations. In *Proc. of WEASELTech*, 2007.

[5] S. Mukherjea and J. T. Stasko. Toward visual debugging: integrating algorithm animation capabilities within a source-level debugger. *ACM Trans. Comput.-Hum. Interact.*, 1(3), 1994.

[6] G. Rothermel, R. H. Untch, C. Chu, and M. J. Harrold. Prioritizing test cases for regression testing. *IEEE Trans. on Soft. Engin.*, 27(10):929–948, 2001.

[7] A. M. Smith and G. M. Kapfhammer. An empirical study of incorporating cost into test suite reduction and prioritization. In *Proc. of 24th SAC*, 2009.

Contact Author: Adam M. Smith (ams292@cs.pitt.edu)