Interactive Multiplots For Comparing Coverage Effectiveness

Adam M. Smith* Joshua J. Geiger†

University of Pittsburgh

ABSTRACT

Software testing increases confidence in the correctness of application source code. Using different orderings of a test suite can enable the earlier detection of defects which allows developers to start fixing them sooner. Finding the best ordering is difficult and there are many approaches that prioritize the ordering, resulting in many possible test suite orderings to choose from. This paper presents a tool that allows the user to visually examine the results of several test suite prioritization techniques and to use interaction to gain insights about the properties of the test suite and its different orders.

1 INTRODUCTION

Inevitably, errors are made while designing and implementing software systems. Software developers execute tests $\{t_1, t_2, t_3, \ldots, t_n\}$ in a test suite T to isolate defects and gain confidence in the correctness of the code. Each test case in the test suite exercises specific points in the system, comparing the actual output of the code to the hand computed expected one. If a test fails, then a defect is present in the area that the test executes. As the source code grows in size and number of features, new tests are written for the new functionality. To ensure that the new features do not cause the system to regress, developers include every previously written test in the collection of tests. This process of executing and re-executing the entire test suite is known as regression testing.

Gradually, adding new tests and retaining old ones increases the size of the test suite until its execution time can become prohibitively expensive. In some cases the regression test suite runs for several weeks [4]. One method of altering the test suite to resolve this issue is test suite prioritization [8]. Test suite prioritization attempts to find a reordering of the test cases that is more likely to locate defaults earlier in the execution of the test suite without risking the loss of coverage by removing a test. The tests are ordered based on certain criteria that are obtained during a process called coverage monitoring.

Coverage monitoring measures the code coverage of a test case t_i. Code coverage describes any metric that enumerates specific points in the source code that are executed when a test is run, whether they are a line, a block, a method, a branch [10], call trees [**?**] ,or some other type. Each specific program point is called a requirement. Given a test suite T, coverage monitoring gives a set of requirements $R(T) = \langle r_1, r_2, \ldots, r_m \rangle$. Each individual test t_i is associated with a subset of requirements $R(t_i) \subseteq R(T)$, which it is said to *cover*.

Figure 1 shows an example of a test suite with 4 tests and 5 requirements. An X in a cell represents that the test for that row covers the requirement in that column. Imagine that the shown test suite is run in its original order. In this case all of the requirements are not covered until 8 time units have passed. Conversely, if the test suite is executed in reverse order all of the requirements are covered in 4 time units. Covering all of the requirements sooner allows for a higher chance to find faults earlier so that the developer can more quickly begin to make changes.

	r_1	r_2	r_3	r_4	r_5	Execution Time
t_1	X	X	X	X		4
t_2			X	X		1
t_3		X				1
t_4	X				X	2

Figure 1: Example Test Suite

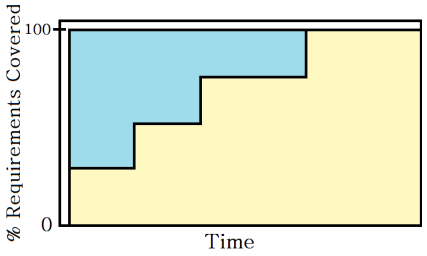

Figure 2: Coverage Effectiveness

The metric coverage effectiveness (CE) [7] was developed in order to rate a test suite prioritization. CE is derived from the step function of the cumulative coverage of a test suite as shown in Figure 1. Each test suite offers the possibility of covering more total requirements. When the cumulative coverage is plotted against time a step function is formed. CE itself is calculated by dividing the area under the step function for the actual order by the area under the curve of a test suite that covers all of its requirements instantly. This value is between 0 and 1 where 0 means that no requirements were covered, and 1 means that all of the requirements were covered instantly.

2 MOTIVATION

There are a total of $n!$ possible orderings of a test suite where n is the number of tests. Clearly for most test suites these cannot all be enumerated to find the highest CE. For this reason there are several algorithmic ways to find good orderings, including random sampling. However, it is difficult to interpret the results of several algorithms or random samples by reading text alone. Therefore, this project aimed to create a visualization that can allow software testers to instantly see how algorithmic prioritizers perform and also to examine a set of random prioritizations simultaneously. There is related work in the field of visualizing test suites [5] [3], however, they focus on fault localization or different features of a test suite.

Examining the CE functions of several reorderings for a given test suite will reveal the effectiveness of the different prioritization techniques on that suite. Figure 2 shows a multiplot of CE functions for 50 random prioritizations. This static image does not allow for easy identification of the source of the prioritization or limiting the visualization to only prioritizations of interest. Also, the static image cannot display qualitative data about each prioritization. To allow for more functionality and information communication an interactive tool was developed instead of only using static images to

*e-mail: ams292@cs.pitt.edu

†e-mail:jj55@cs.pitt.edu

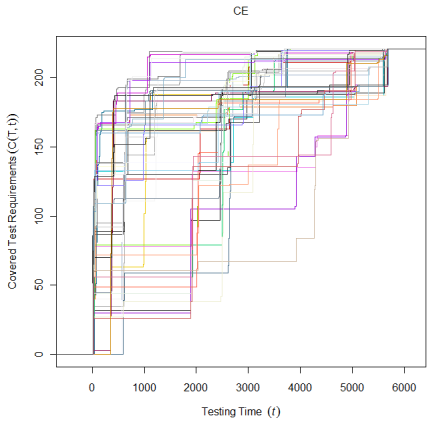

Figure 3: Coverage Effectiveness Multiplot

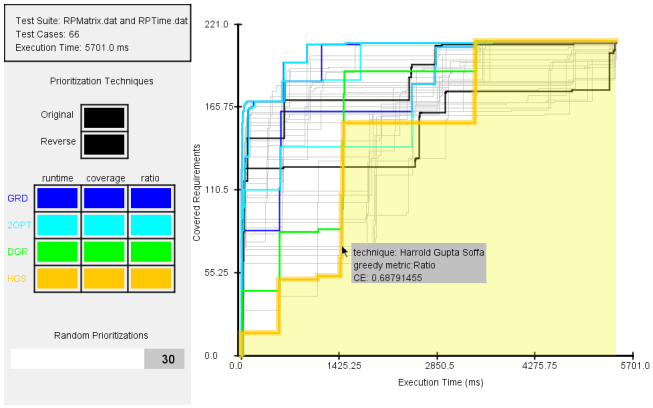

Figure 4: Interactive Multiplot

visualize CE multiplots.

3 Tool Design

Our tool is designed to aid in analyzing multiple prioritizations of a test suite through user interaction of CE multiplots. By providing the tool with coverage and timing data for a test suite, several prioritizations are generated using the techniques described by Smith and Kapfhammer [9]. Due to the nature of step functions consisting of only vertical and horizontal lines, plotting several functions on a single graph may obscure lines, thus making it difficult to track and compare prioritizations. A solution to this is pruning the graph to only the plots in which the user is interested. However, generating a new static multiplot for each comparison is cumbersome. Therefore, in a similar approach to Becker et al. [2] and a NY Times interactive visualization of market statstics [1] the tool allows users to interactively select techniques displayed and directly manipulate the corresponding plot.

Figure 4 provides a screen capture of the tool showing a test suite's prioritizations. The left panel provides data selection, while the right panel displays the multiplot. The upper left corner gives general information on the test suite. In this image, all techniques are currently selected to appear in the multiplot. This is controlled by the grid of color-coded buttons which allow the user to select/deselect the prioritizations displayed. Additionally, a mouseover of the lines of individual functions will highlight the line and shade the area under the curve, as well as, provide a label identfying the technique, parameters to the technique, and the CE value of the prioritization represented by the line. This is superior to a static graph since the user immediately knows which prioritization the line corresponds to as well as the quality of the prioritization as determined by its CE value. A slider on the lower left corner allows the user to generate varying numbers of random prioritizations that appear in the multiplot as thin, light gray lines.

4 Evaluation

Feedback on the usability and effectiveness in aiding CE comparisons of the tool was primarily provided by Dr. Gregory M. Kapfhammer, an expert in software engineering, software testing and analysis, and computer systems [6]. Response to the tool was generally positive, the evaluator found the tool useful in displaying the information but made suggestions on areas where the tool could help to interpret the data. Some suggestions include:

- Displaying coverage density information
- Relate 'goodness' of approach (CE) through glyphs or colors
- Empirical cumulative distribution functions of the random

A few other suggestions on layout were also made, but our evaluator stated that the tool was already useful enough that he would use it in his own research, as well as, utilize it in teaching his future software engineering courses. Overall, the tool met its goal of providing simple interactive technique for comparing prioritization effectiveness.

Acknowledgements

The authors wish to thank Gregory Kapfhammer, Manos Renieris, and Liz Marai for assistance with this project.

References

[1] X. G. Amanda Cox and D. Leonhard. How This Bear Market Compares. `http://www.nytimes.com/interactive/2008/10/11/business/20081011_BEAR_MARKETS.html`.

[2] R. A. Becker, S. G. Eick, and A. R. Wilks. Visualizing Network Data. *IEEE Transactions on Visualization and Computer Graphics*, 1:16–28, 1995.

[3] M. Breugelmans and B. Van Rompaey. Testq: Exploring structural and maintenance characteristics of unit test suites.

[4] S. G. Elbaum, A. G. Malishevsky, and G. Rothermel. Test Case Prioritization: A Family of Empirical Studies. *IEEE Transactions on Software Engineering*, 28(2):159–182, 2002.

[5] J. A. Jones, M. J. Harrold, and J. Stasko. Visualization of test information to assist fault localization. In *ICSE '02: Proceedings of the 24th International Conference on Software Engineering*, pages 467–477, New York, NY, USA, 2002. ACM.

[6] G. M. Kapfhammer. Gregory M. Kapfhammer's Homepage. `http://www.cs.allegheny.edu/~gkapfham`.

[7] G. M. Kapfhammer and M. L. Soffa. Using coverage effectiveness to evaluate test suite prioritizations. In *Proceedings of the ACM Workshop on Empirical Assessment of Software Engineering Languages and Technologies*, November 2007.

[8] G. Rothermel, R. H. Untch, C. Chu, and M. J. Harrold. Prioritizing Test Cases For Regression Testing. *IEEE Transactions on Software Engineering*, 27(10):929–948, 2001.

[9] A. M. Smith and G. M. Kapfhammer. An empirical study of incorporating cost into test suite reduction and prioritization. In *SAC '09: Proceedings of the 2009 ACM symposium on Applied Computing*, pages 461–467, New York, NY, USA, 2009. ACM.

[10] H. Zhu, P. A. V. Hall, and J. H. R. May. Software unit test coverage and adequacy. *ACM Comput. Surv.*, 29(4):366–427, 1997.