Using Coverage Effectiveness to Evaluate Test Suite Prioritizations

Gregory M. Kapfhammer
Department of Computer Science
Allegheny College
gkapfham@allegheny.edu

Mary Lou Soffa
Department of Computer Science
University of Virginia
soffa@cs.virginia.edu

ABSTRACT

Regression test suite prioritization techniques reorder a test suite with the goal of ensuring that the reorganized test suite finds faults faster than the initial ordering. It is challenging to empirically evaluate the effectiveness of a new test case arrangement because existing metrics (i) require fault seeding or (ii) ignore test case costs. This paper presents a coverage effectiveness (CE) metric that (i) obviates the need to seed faults into the program under test and (ii) incorporates available data about test case execution times. A test suite is awarded a high CE value when it quickly covers the test requirements. It is possible to calculate coverage effectiveness regardless of the coverage criterion that is chosen to evaluate test case quality. The availability of an open source CE calculator enables future case studies and controlled experiments to use coverage effectiveness when evaluating different approaches to test suite prioritization.

Categories and Subject Descriptors: D.2.5 [Software Engineering]: Testing and Debugging

General Terms: Verification, Experimentation

Keywords: test suite prioritization, coverage effectiveness

1. INTRODUCTION

Regression testing techniques ensure that the addition of bug fixes or new functionality does not negatively impact the correctness of a software application. Since regression testing may be prohibitively expensive in terms of test case execution time [3], prioritization approaches reorder a test suite so that the highest priority tests are executed earlier in the testing process. With the ultimate goal of finding program faults faster, a prioritization method could reorganize a test suite so that the tests rapidly achieve complete coverage of the test requirements. Beyond evaluating the efficiency of the prioritization algorithm, it is also important to measure the effectiveness of the reordered test suite created by the prioritizer.

Since both researchers and practitioners must assess the usefulness of a prioritization technique, the software engineering community recently developed several measures of effectiveness. Rothermel et al. and Saff et al. respectively proposed the *average percentage of faults detected* (APFD) [3] and the *time to failure* (TTF) [4] as two metrics for empirically evaluating a prioritized test suite. APFD classifies a test ordering as highly effective if it rapidly exposes the faults that were seeded into the program under test. According to the TTF metric, a test prioritization is highly effective if it detects the first fault in a short amount of time. While the seeded defects may be chosen from the known defects in the program under test, these metrics can also be calculated after using a mutation testing tool to insert synthetic faults. If a mutation tester is not available, then it is also possible to manually seed faults into the source code of an application. Yet, reliance upon manual fault seeding can be problematic because it is costly, prone to error, and a potential source for experimental bias.

Since high coverage test cases are more likely to reveal program faults than those with low coverage [1], it is sensible to reorder test suites in a manner that improves the rate of requirement coverage. As an alternative to fault-based metrics such as APFD and TTF, Li et al. use coverage-based metrics such as the *average percentage of blocks covered* (APBC) [2]. APBC identifies a test suite ordering as highly effective if it quickly covers the basic blocks within the program under test. Even though APBC obviates the need for fault seeding, it ignores the costs associated with executing a test case and thus it may inaccurately characterize effectiveness. This paper describes a *coverage effectiveness* (CE) metric that fully incorporates both the cost and the coverage of each test case. CE is useful when manual fault seeding is too costly or mutation testing is not feasible due to tool unavailability. In order to better control the threats to construct validity, we advocate the use of CE in conjunction with previously developed metrics such as APFD, TTF, and APBC.

The calculation of CE requires the creation of a *cumulative coverage* function that describes how requirement coverage varies over time. We construct this coverage function under the assumption that a requirement is marked as covered upon the completion of one of its covering tests. Integrating this function yields the *coverage area* of the test suite. Intuitively, a large coverage area suggests that a particular test suite reordering is highly effective. In support of an approach to comparing different prioritizations of the same test suite, we define coverage effectiveness as the ratio between the reordered suite's coverage area and the coverage area of the ideal test suite that immediately covers all requirements. As defined, a CE value falls inclusively between 0 and 1 with a high value indicating a good test suite.

WEASELTech'07, November 6, 2007, Atlanta, Georgia, USA.

Test Case	Cost (sec)	Requirements				
		R_1	R_2	R_3	R_4	R_5
T_1	5	✓	✓			
T_2	10	✓	✓	✓		✓
T_3	4	✓			✓	✓

Total Testing Time = 19 seconds

Figure 1: Characteristics of a Test Suite.

Ordering	CE	CE_u
$T_1\ T_2\ T_3$	.3789	.4
$T_1\ T_3\ T_2$	.5053	.4
$T_2\ T_1\ T_3$	.3789	.5333
$T_2\ T_3\ T_1$	.4316	.6
$T_3\ T_1\ T_2$	.5789	.4557
$T_3\ T_2\ T_1$	.5789	.5333

Figure 2: Coverage Effectiveness Values.

2. FORMULATING THE METRIC

When provided with cost and coverage information, as depicted in Figure 1, it is possible to calculate the coverage effectiveness of test suite T, denoted $\mathrm{CE}(T)$. If test cost information is not available, then we can assume that each test consumes a single unit of time and subsequently compute $\mathrm{CE_u}(T)$, the unit coverage effectiveness of T. For example, suppose that a test suite $T = \langle T_1, T_2, T_3\rangle$ and it covers a total of five requirements while testing program P. These requirements might be (i) nodes and edges in the program's control flow graph, (ii) definition-use associations, or (iii) call tree paths. Figure 1 characterizes test suite T according to execution time and requirement coverage (e.g., test T_2 takes ten seconds to execute while T_1 and T_3 respectively consume five and four seconds).

Figure 2 furnishes the CE and $\mathrm{CE_u}$ values for the $3! = 3\times2\times1 = 6$ different orderings of test suite T. This example demonstrates that the inclusion of test case execution costs does impact the measurement of effectiveness. For instance, $\mathrm{CE_u}$ equivalently ranks the orderings $O_1 = \langle T_1, T_2, T_3\rangle$ and $O_2 = \langle T_1, T_3, T_2\rangle$ while CE classifies the latter as more effective. This result is due to the fact that O_2's first two tests cover four requirements in nine seconds while the corresponding tests in O_1 take fifteen seconds to cover four requirements. For a given test order, CE may be higher than $\mathrm{CE_u}$ or vice-versa. For example, $\langle T_2, T_1, T_3\rangle$ results in a low value for CE and a high $\mathrm{CE_u}$ score because CE incorporates the substantial cost of running T_2 and $\mathrm{CE_u}$ assumes that T_2's running time is equivalent to the other test cases.

Suppose that a testing tool creates test suites T' and T'' after applying two different prioritization techniques to the original test suite T. If $\mathrm{CE}(T') > \mathrm{CE}(T'')$, then we know that T' is more coverage effective than T'' and thus we would prefer the first approach to prioritization instead of the second. Equation (1) defines $\mathrm{CE}(T) \in [0,1]$ such that the numerator is the integral of $C(T,t)$, the requirement coverage for test suite T at time t during test execution. Equation (1) shows that the denominator is the integral of $\bar{C}(T,t)$, the requirement coverage for the ideal version of T. Calculating $\mathrm{CE}(T)$ requires the integration of both the C and $\bar{C}$ functions in the closed interval between 0 and $t(n)$. Equation (2) defines $t(m)$ as the time required to execute the first m test cases in T. For example, $t(1)$ would return the time overhead for test T_1 and $t(n)$ calculates the running time for the entire suite of n test cases. Finally, Equation (1) uses t to stand for a specific point in time during testing.

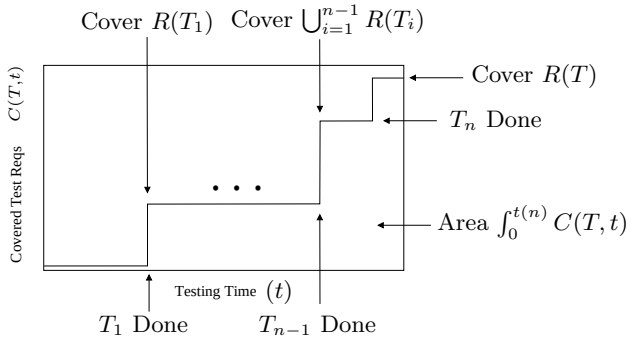

Figure 3: The Cumulative Coverage of a Test Suite.

$$\mathrm{CE}(T) = \frac{\int_0^{t(n)} C(T,t)}{\int_0^{t(n)} \bar{C}(T,t)} \quad (1) \qquad t(m) = \sum_{i=1}^{m} time(T_i) \quad (2)$$

If T covers the requirements in the set $R(T)$, then $\bar{C}(T,t)$ in Equation (3) shows that an ideal suite would immediately cover all $R_j \in R(T)$. Equation (4) defines the step function $C(T,t)$ that describes the cumulative coverage of T at time t during testing. We define $C(T,t)$ as an $(n+1)$-part piecewise function when $T = \langle T_1, \ldots, T_n\rangle$. Equation (4) reveals that $C(T,t) = 0$ until the completion of test case T_1 (i.e., $t < t(1)$). In the time period after the execution of T_1 and during the running of T_2 (i.e., $t \in [t(1), t(2))$), the value of C shows that T has covered a total of $|R(T_1)|$ requirements. The function C maintains the maximum height of $|R(T)|$ for all time points $t \geq t(n)$, as graphically depicted in Figure 3. $\mathrm{CE_u}(T)$ is formulated in a similar manner to Equations (1) through (4), except that $t(i) = i$ for all T_i.

$$\bar{C}(T,t) = \left|\bigcup_{i=1}^{n} R(T_i)\right| \quad (3)$$

$$C(T,t) = \begin{cases} 0 & t < t(1) \\ |R(T_1)| & t \in [t(1), t(2)) \\ \vdots & \vdots \\ \left|\bigcup_{i=1}^{n-1} R(T_i)\right| & t \in [t(n-1), t(n)) \\ |R(T)| & t \geq t(n) \end{cases} \quad (4)$$

In summary, we recommend that future empirical studies use the CE and $\mathrm{CE_u}$ metrics to evaluate different prioritization techniques. As part of future research, we intend to investigate search-based techniques for identifying prioritizations with high CE values. An open source CE calculator and additional details about the metrics are available at:

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

3. REFERENCES

[1] M. Hutchins, H. Foster, T. Goradia, and T. Ostrand. Experiments of the effectiveness of dataflow- and controlflow-based test adequacy criteria. In *Proc of 16th ICSE*, pages 191–200, 1994.

[2] Z. Li, M. Harman, and R. Hierons. Search algorithms for regression test case prioritization. *IEEE Trans. on Soft. Engin.*, 33(4):225–237, 2007.

[3] G. Rothermel, R. J. Untch, and C. Chu. Prioritizing test cases for regression testing. *IEEE Trans. on Softw. Eng.*, 27(10):929–948, 2001.

[4] D. Saff, S. Artzi, J. H. Perkins, and M. D. Ernst. Automatic test factoring for Java. In *Proc. of 20th ASE*, 2005.