Incorporating Varying Test Costs and Fault Severities into Test Case Prioritization

Sebastian Elbaum
Department of Computer Science and Engineering
University of Nebraska-Lincoln
Lincoln, Nebraska
elbaum@cse.unl.edu

Alexey Malishevsky
Computer Science Department
Oregon State University
Corvallis, OR
malishal@cs.orst.edu

Gregg Rothermel
Computer Science Department
Oregon State University
Corvallis, OR
grother@cs.orst.edu

ABSTRACT
Test case prioritization techniques schedule test cases for regression testing in an order that increases their ability to meet some performance goal. One performance goal, *rate of fault detection*, measures how quickly faults are detected within the testing process. In previous work we provided a metric, APFD, for measuring rate of fault detection, and techniques for prioritizing test cases to improve APFD, and reported the results of experiments using those techniques. This metric and these techniques, however, applied only in cases in which test costs and fault severity are uniform. In this paper, we present a new metric for assessing the rate of fault detection of prioritized test cases, that incorporates varying test case and fault costs. We present the results of a case study illustrating the application of the metric. This study raises several practical questions that might arise in applying test case prioritization; we discuss how practitioners could go about answering these questions.

Keywords
test case prioritization, regression testing, test cost, fault severity, rate of fault detection

1 INTRODUCTION

Software engineers often save the test suites they develop so that they can reuse those test suites later during regression testing. Reusing all of the test cases in a test suite, however, can be expensive: for example, one of our industrial collaborators reports that for one of its products of about 20,000 lines of code, the entire test suite requires seven weeks to run. In cases such as this, testers may want to order their test cases such that those with the highest priority, according to some criterion, are run earlier than those with lower priority.

Test case prioritization techniques [2, 11, 13] schedule test cases in an order that increases their effectiveness at meeting some performance goal. For example, test cases might be scheduled in an order that achieves code coverage as quickly as possible, exercises features in order of frequency of use, or reflects their historically observed abilities to detect faults.

One potential goal of test case prioritization is to increase a test suite's *rate of fault detection* – that is, how quickly that test suite detects faults during the testing process. An increased rate of fault detection during testing provides earlier feedback on the system under test, allowing debugging to begin earlier, and supporting faster strategic decisions about release schedules. Further, an improved rate of fault detection can increase the likelihood that if the testing period is cut short, test cases that offer the greatest fault detection ability in the available testing time will have been executed.

In previous work [2, 11] we provided a metric, APFD, which measures the average cumulative percentage of faults detected over the course of executing the test cases in a test suite in a given order. We showed how the APFD metric can be used to quantify and compare the rates of fault detection of test suites. We presented several techniques for prioritizing test cases to improve APFD during regression testing, and empirically evaluated their effectiveness. Our results indicated that several of the techniques can improve APFD, and that this improvement can occur even for the least sophisticated (and least expensive) techniques.

Although successful in application to the class of problems for which they were designed, the APFD metric and techniques relied on the assumption that test costs and fault severities are uniform. In practice, however, test costs and fault severities can vary widely. When this occurs, the APFD metric can assign inappropriate values to test case orders, and techniques designed to improve test case orders under that metric can produce unsatisfactory orders.

Therefore, in this paper we present a new, more general metric for measuring rate of fault detection that accounts for varying test case and fault costs. To further investigate this metric and its application, we present prioritization techniques that account for these varying costs, and results of a case study in which we apply these techniques under several different test case and fault severity cost distributions. This study raises several practical questions that might arise in applying test case prioritization, and we discuss how practitioners could go about answering them.

0-7695-1050-7/01 $10.00

2 THE TEST CASE PRIORITIZATION PROBLEM

In [2, 11] we described the test case prioritization problem; we briefly review that work here.

The Test Case Prioritization Problem:

Given: T, a test suite; PT, the set of permutations of T; f, a function from PT to the real numbers.

Problem: Find $T' \in PT$ such that $(\forall T'')\ (T'' \in PT)\ (T'' \neq T')\ [f(T') \geq f(T'')]$.

In this definition, PT is the set of possible prioritizations (orders) of T, and f is an objective function that, applied to any such order, yields an *award value* for that order.

There are many possible goals for prioritization. For example, testers may wish to increase the coverage of code in the system under test at a faster rate, increase their confidence in the reliability of the system at a faster rate, or increase the rate at which test suites detect faults in that system during regression testing. In the definition of the test case prioritization problem, f represents a quantification of such a goal.

Given any prioritization goal, various *test case prioritization techniques* may be used to meet that goal. For example, to increase the rate of fault detection of test suites, we might prioritize test cases in terms of the extent to which they execute modules that have tended to fail in the past. Alternatively, we might prioritize test cases in terms of greatest-to-least coverage-per-cost of code components, or in terms of greatest-to-least coverage-per-cost of features listed in a requirements specification. In any case, the intent behind the choice of a prioritization technique is to increase the likelihood that the prioritized test suite can better meet the goal than would an ad hoc or random order of test cases.

3 MEASURING EFFECTIVENESS

To measure how rapidly a prioritized test suite detects faults (the rate of fault detection of the test suite) we require an appropriate objective function f. For this purpose, in [2, 11] we defined a metric, APFD, which represents the weighted average of the percentage of faults detected during the execution of the test suite. APFD values range from 0 to 100; higher values imply faster (better) fault detection rates.

Consider an example program with 10 faults and a test suite of 5 test cases, **A** through **E**, with fault detecting abilities as shown in Figure 1.A. Suppose we place the test cases in order **A–B–C–D–E** to form prioritized test suite $T1$. Figure 1.B shows the percentage of detected faults versus the fraction of $T1$ used. After running test case **A**, 2 of the 10 faults are detected; thus 20% of the faults have been detected after 0.2 of $T1$ has been used. After running test case **B**, 2 more faults are detected and thus 40% of the faults have been detected after 0.4 of the $T1$ has been used. The area under the curve represents the weighted average of the percentage of faults detected over the life of the test suite. This area is the prioritized test suite's average percentage faults detected metric (APFD); the APFD is 50% in this example.

Figure 1.C reflects what happens when the order of test cases is changed to **E–D–C–B–A**, yielding a "faster detecting" suite than $T1$ with APFD 64%. Figure 1.D shows the effects of using a prioritized test suite $T3$ whose test case order is **C–E–B–A–D**. By inspection, it is clear that this order results in the earliest detection of the most faults and illustrates an optimal order, with APFD 84%.

Limitations of the APFD Metric

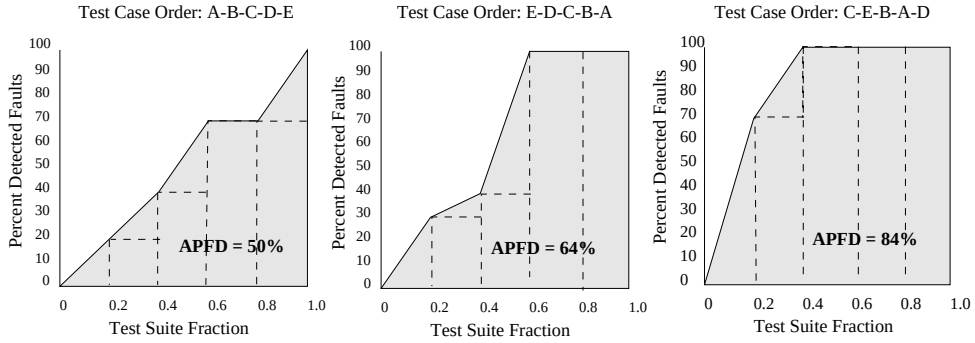
The APFD metric just presented relies on two assumptions: (1) all faults have equal severity, and (2) all test cases have equal costs. (These assumptions are manifested in the fact that the metric simply plots the percentage of faults detected against the fraction of the test suite run.) Our previous empirical results [2, 11] suggest that when these assumptions hold, the metric operates well. In practice, however, there are cases in which these assumptions do not hold: cases in which faults vary in severity and test cases vary in cost. In such cases, the APFD metric can provide unsatisfactory results, as the following simple examples illustrate.

Example 1. Consider the testing scenario illustrated in Figure 1. Under the APFD metric, when all ten faults are equally severe and all five test cases are equally costly, orders **A–B–C–D–E** and **B–A–C–D–E** are equivalent in terms of rate of fault detection; swapping **A** and **B** alters the rate at which *particular* faults are detected, but not the overall rates of fault detection. This equivalence is reflected in equivalent APFDs (50%). Suppose, however, that **B** is twice as costly as **A**; requiring two hours to execute where **A** requires one. In terms of faults-detected-per-hour, test case order **A–B–C–D–E** is preferable to order **B–A–C–D–E**, resulting in faster detection of faults. The APFD metric, however, does not distinguish between the two orders.

Example 2. Again working with the scenario given in Figure 1, suppose that all five test cases have equivalent costs, and suppose that faults 2-10 have severity k, while fault 1 has severity $2k$. In this case, test case **A** detects this more severe fault along with one less severe fault, whereas test case **B** detects only two less severe faults. In terms of fault-severity-detected, test case order **A–B–C–D–E** is preferable to order **B–A–C–D–E**. Again, the APFD metric does not distinguish between these two orders.

Example 3. Examples 1 and 2 provide cases in which the APFD metric proclaims two orders *equivalent* where our intuitions say they are not. It is also possible, when test costs or fault severities differ, for the APFD metric to assign a *higher* value to a test case order that we would consider *less* valuable. Working again with Figure 1, suppose that all ten faults are equally severe, and that test cases **A**, **B**, **D**, and **E** each require one hour to execute, but test case **C** requires ten hours. Consider test case order **C–E–B–A–D**. Under the APFD metric this order is assigned an APFD value of 84% (see Figure 1.D). Consider alternative test case order **E–C–B–A–D**. This order is illustrated with respect to the APFD

test	fault									
	1	2	3	4	5	6	7	8	9	10
A	x				x					
B						x	x			
C	x	x	x	x	x	x	x			
D					x					
E								x	x	x



A. Test suite and faults exposed **B. APFD for prioritized suite T1** **C. APFD for prioritized suite T2** **D. APFD for prioritized suite T3**

Figure 1: Examples illustrating the APFD metric.

metric in Figure 2; because that metric does not differentiate test cases in terms of relative costs, all vertical bars in the graph (representing individual test cases) have the same width. The APFD for this order is 76% – lower than the score for test case order **C–E–B–A–D**. However, in terms of faults-detected-per-hour, the second order (**E–C–B–A–D**) is preferable: it detects 3 faults in the first hour, and remains better in terms of faults-detected-per-hour than the first order up through the end of execution of the second test case.

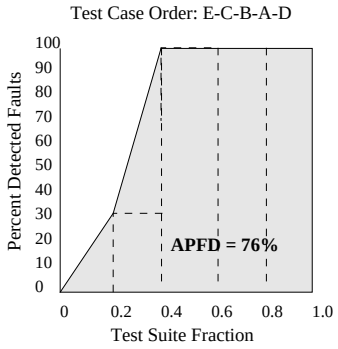


Figure 2: APFD for Example 3.

An analogous example can be created by using varying fault severities while holding test case costs uniform.

Example 4. Finally, consider an example in which both fault severities and test case costs vary. Suppose that test case **B** is twice as costly as test case **A**, requiring two hours to execute where **A** requires one. In this case, in Example 1, assuming that all ten faults were equally severe, we found test case order **A–B–C–D–E** preferable. However, if the faults detected by **B** are more costly than the faults detected by **A**, order **B–A–C–D–E** may be preferable. For example, suppose test case **A** has cost "1", and test case **B** has cost "2". If faults 1 and 5 (the faults detected by **A**) are assigned severity "1", and faults 6 and 7 (the faults detected by **B**) are assigned severities greater than "2", then order **B–A–C–D–E** achieves greater "units-of-fault-severity-detected-per-unit-test-cost" than does order **A–B–C–D–E**.[1] Again, the APFD metric would not make this distinction.

A New Cost Model

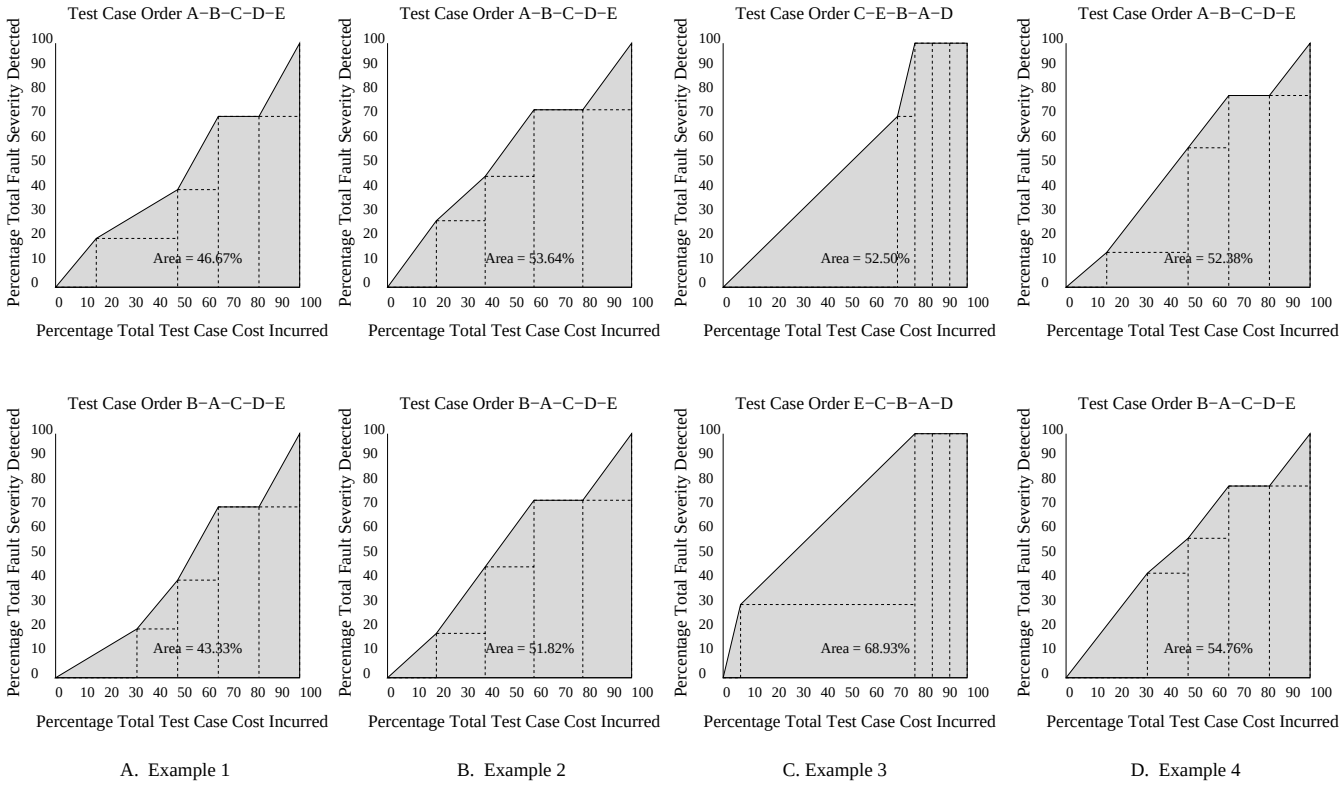
The foregoing examples suggest that, when considering the relative merits of test cases, a measure for rate-of-fault-detection that assumes that test case costs and fault severities are uniform can produce unsatisfactory results.

The notion that a tradeoff exists between the costs of testing and the costs of leaving undetected faults in software is fundamental in practice, and testers face and make decisions about this tradeoff frequently. It is thus appropriate that this tradeoff be considered when prioritizing test cases. A metric for evaluating test case orders must accommodate the factors underlying this tradeoff. It is our thesis that such a metric should *reward test case orders proportionally to their rate of "units-of-fault-severity-detected-per-unit-test-cost"*.

We have created such a metric by adapting our APFD metric; we call our new "cost-cognizant" metric $APFD_C$. In terms of the graphs used in Figures 1 and 2, creation of this new metric entails two modifications. First, instead of letting the horizontal axis in such a graph denote "Test Suite Fraction", we let it denote "Percentage Total Test Case Cost Incurred". Now, each test case in the test suite is represented by an interval along the horizontal axis, whose length is proportional to the percentage of total test suite cost accounted for by that test case. Second, instead of letting the vertical axis in such a graph denote "Percent Detected Faults", we let it denote "Percentage Total Fault Severity Detected". Now, each fault detected by the test suite is represented by an interval along the vertical axis, whose height is proportional to the percentage of total fault severity that fault accounts for.

In this context, the "cost" of a test case and the "severity" of a fault can be measured in various ways. If time (for test

[1] In this example we have assumed, for simplicity, that a "unit" of fault severity is equivalent to a "unit" of test case cost. Clearly, in practice, the relationship between fault severity and test case cost will vary among applications, and quantifying this relationship may be non-trivial. We discuss this further in the following sections.



A. Example 1 B. Example 2 C. Example 3 D. Example 4

Figure 3: Examples illustrating the APFD$_C$ metric.

execution, setup, and validation) is the primary component of test case cost then execution adequately tracks that cost. However, practitioners could also assign test case costs based on factors such as hardware costs or engineers' salaries.

Similarly, fault severity could be measured in terms of the time required to locate and correct a fault, or practitioners could factor in the costs of lost business, damage to persons or property, and so forth. In any case, the APFD$_C$ metric supports these interpretations. (In the context of APFD$_C$ we are concerned not with predicting these costs, which may be difficult, but with measuring the costs after they have occurred, so as to evaluate various test case orders.)

Under this new interpretation, in graphs such as those just discussed, a test case's contribution is "weighted" along the horizontal dimension in terms of test case cost, and along the vertical dimension in terms of the cumulative severity of faults it reveals. In such graphs, the curve delimits a greater area for a test case order that exhibits greater "units-of-fault-severity-detected-per-unit-test-cost"; the area delimited constitute our new APFD$_C$ metric.

To illustrate the APFD$_C$ metric from this graphical point of view, Figure 3 presents graphs for each of the four examples presented in the preceding subsection. The leftmost pair of graphs (Figure 3.A) correspond to Example 1: the upper of the pair represents the APFD$_C$ for test case order **A–B–C–D–E**, and the lower represents the APFD$_C$ for order **B–A–C–D–E**. Note that whereas the (original) APFD metric did not distinguish the two orders, the APFD$_C$ metric gives preference to the faster-detecting order, **A–B–C–D–E**.

The other pairs of graphs illustrate the application of the APFD$_C$ metric in Examples 2, 3, and 4, The pair of graphs in Figure 3.B, corresponding to Example 2, show how the new metric gives a higher award to the test case order that reveals the more severe fault earlier (**A–B–C–D–E**), under the assumption that the severity value assigned to faults 2-10 is 1 and the severity value assigned to fault 1 is 2. The pair of graphs in Figure 3.C, corresponding to Example 3, show how the new metric distinguishes test case orders involving a high-cost test case **C**: instead of undervaluing order **E–C–B–A–D**, the metric now assigns it greater value than order **C–E–B–A–D**. Finally, the pair of graphs in Figure 3.D, corresponding to Example 4, show how the new metric distinguishes between test case orders when both test case costs and fault severities are non-uniform, under the assumptions that test case B has cost 2 while all other test cases have cost 1, and that faults 6 and 7 have severity 3 while all other faults have severity 1. In this case, the new metric assigns a greater value to order **B–A–C–D–E** than to order **A–B–C–D–E**.

Our APFD$_C$ metric can be quantitatively described as follows. Let T be a test suite containing n test cases with costs $t_1, t_2, \ldots, t_n$. Let F be a set of m faults revealed by T, and let $f_1, f_2, \ldots, f_m$ be the severities of those faults. Let TF_i be the first test case in an ordering T' of T that reveals fault i.

The (cost-cognizant) weighted average percentage of faults detected during the execution of T' is given by the equation:

$$\text{APFD}_C = \frac{\sum_{i=1}^{m}(f_i \times (\sum_{j=TF_i}^{n} t_j - \frac{1}{2}t_{TF_i}))}{\sum_{i=1}^{n} t_i \times \sum_{i=1}^{m} f_i} \quad (1)$$

Note that Equation 1 remains applicable when either test case costs or fault severities are identical. Further, when both test case costs and fault severities are identical, the formula reduces to the formula for APFD.

Estimating Test Case and Fault Severity Costs

As mentioned earlier, the APFD_C metric is concerned not with predicting test case costs and fault severities, but with assessing prioritization orders after-the-fact when these values are known. As such, the metric serves to evaluate the results of prioritization, and to compare and evaluate prioritization techniques. When creating prioritization techniques themselves, however, if we wish those techniques to account for varying test case and fault severity costs, we need to be able to predict such costs.

Where test case costs are concerned, when prioritizing for regression testing, prioritization techniques can take advantage of data gathered in previous executions of the test suite (e.g., timestamps in test script outputs). Of course, this data may not reflect precisely the test case costs that will occur in the future after the program under test has been modified, so we must be satisfied with estimates. It seems reasonable, however, to expect that historical test case cost data can be used to predict future test case costs with sufficient accuracy. Finally, in practice, such estimates are often available [10].

Predicting fault severity in practice is more difficult. Several techniques, however, have been proposed for use in doing so [8, 10, 12]. For example, Musa [8] introduces guidelines for the definition of failure severity classes by which each system operation can be associated with a class of failures, usually based on experience and historical data. A similar procedure can be used to estimate module criticality. More consistent classifications arise from software with high safety requirements, where each failure event must have a probability and a risk level (e.g, catastrophic, critical, and negligible). The criticality of each component is then derived from its association with a set of failure events [12].

Thus, even if fine-grained estimates of fault severities (or, for that matter, test case costs) cannot be obtained, it may be useful to identify coarse-grained ranking schemes by assigning numeric cost values to categories of severities (or test costs). In Section 4, we describe two test case cost ranking schemes and one fault severity ranking scheme based on industry data, and we use them in a case study of prioritization.

Test Case Prioritization Techniques

In previous work [2, 11] we have considered various prioritization techniques; in this paper, because our primary focus is on metrics for measuring rate of fault detection, we restrict our attention to four of these. One technique is an experimental control, and three are practical heuristics. All three heuristics have previously been investigated in forms that do not account for test case costs and fault severity; the versions that we present here are adapted to account for these effects. We briefly describe each technique and the necessary adaptations below, additional details can be found in [6].

Random Ordering (random).
As an experimental control, one prioritization "technique" that we consider is to randomly order the test cases in the test suite. Such an ordering makes no attempt to account for varying test case costs and fault severities.

Additional Statement Coverage Prioritization (st-addtl).
Additional statement coverage prioritization greedily selects a test case that yields the greatest statement coverage, then adjusts the coverage data about remaining test cases to indicate their coverage of statements not yet covered, and then repeats this process until all statements covered by at least one test case have been covered. If test cases remain, the process is repeated.

To adapt this technique to the case in which test case costs and fault severities vary, instead of summing the number of statements covered additionally by a test case j to calculate the worth of j, we sum the values of $\frac{\text{criticality}_i}{\text{cost}_j}$ for each statement i covered additionally by j, where criticality_i is an estimate of the severity of a fault occurring in statement i, and cost_j is the cost of test case j. The notion behind this ratio of criticality to cost is to reward test cases that have greater ratios of fault-severity-detected per unit-test-cost.

Additional Function Coverage Prioritization (fn-addtl).
Analogous to additional statement coverage prioritization but operating at the level of functions, this technique prioritizes test cases according to the total number of additional functions they cover. To adapt this technique to the case in which test case costs and fault severities vary, we do as with additional statement coverage, summing $\frac{\text{criticality}_i}{\text{cost}_j}$ values, but in this case, criticality_i is an estimate of the severity of a fault occurring in *function* i, and our sum is across the functions covered by test case j.

Additional Fault Index Prioritization (fn-fi-addtl).
Certain functions are more likely to contain faults than others, and this fault proneness can be associated with measurable software attributes [1, 5, 7]. Additional fault index prioritization takes advantage of this association by prioritizing test cases based on their history of executing fault prone functions. To represent fault proneness, we use *fault indexes* based on principal component analysis [3, 9]. Given these fault indexes, additional fault index prioritization is similar to additional function coverage prioritization. The set of functions that have been covered by previously executed test cases is maintained. To find a next best test case we compute, for each test case, the sum of the fault indexes for each additional function that test case executes. The test case for

which this sum is the greatest wins. This process is repeated until all test cases have been prioritized.

To adapt this technique to handle varying test case costs and fault severities, instead of summing fault index values for each function covered, we sum the values of $\frac{fi_i \times \text{criticality}_\text{i}}{\text{cost}_\text{j}}$ for each function i covered additionally by j, where fi_i is the fault index of i, $\text{criticality}_\text{i}$ is an estimate of the severity of a fault in function i, and cost_j is the cost of test case j.

4 CASE STUDY

To illustrate the practical application of our metric and some of the ramifications of using it, we present the results of a case study. The goal of the case study is to show how different test case cost and fault severity distributions can affect the rate of fault detection as measured by APFD_C.

As a target for this case study we used a program called `space`. `Space` is an interpreter for an array definition language (ADL), developed for the European Space Agency, consisting of 6218 lines of executable C code. We next enumerated the variables that needed to be considered. The independent variables were faulty version, test suite, fault-severity and test-cost distribution, and prioritization technique. The dependent variable was the APFD_C value.

Because the focus of our study was the impact of test case cost and fault severity distributions on prioritization, we attempted to control for the effects of test suite composition and the type of changes represented in each version. For test suites, we used 50 branch-coverage-adequate test suites generated for use in previous studies [11]; these suites ranged in size from 148 to 166 test cases. To control for type of changes we used ten versions of `space` that contained faults of different types; each fault had been discovered during the program's development. The multi-fault versions contained between one and eight faults apiece, averaging 3.3 each.

We used several different test case cost and fault severity distributions. Each distribution was evaluated under each combination of test suite and version, discriminating by prioritization technique. We describe these distributions next.

Test Case Cost and Fault Severity Distributions

We used five test case cost distributions, as follows:

- *Unit:* all test case costs are ones. This corresponds to the case in which test case costs are not considered.
- *Random:* test case costs are uniformly distributed over the range $[1 \ldots 10]$.
- *Normal:* test costs are normally distributed over the range $[1 \ldots 10]$, with mean 5.0, standard deviation 2.0.
- *Mozilla:* Test costs are distributed as in the Mozilla application (see Table 1) into four levels. Mozilla was the original name for Netscape Communicator and is now an open-source web browser involving hundreds of developers and thousands of testers.[2]
- *QTB:* Test costs are distributed as in the QTB application (see Table 2). QTB is a real time embedded software system of more than 300KLOC developed by one of our industrial partners (see Reference [4] for details).

Name	level	description	percentage
HTML	1	least expensive	87
Printing	2	more expensive	1
Smoke tests	3	expensive	2
Buster	4	most expensive	10

Table 1: Mozilla test case cost distribution.

level	description	percentage
1	not expensive	88
10	expensive	12

Table 2: QTB test case cost distribution.

To apply each test case cost distribution (other than unit, whose application was trivial) we generated a set of cost numbers having the required distribution, and then randomly mapped those numbers to test cases.

We used three fault severity distributions, as follows:

- *Unit:* all fault severities are ones. This corresponds to the case in which fault severities are not considered.
- *Mozilla linear:* fault severities are distributed as in the Mozilla application (see Table 3), into six levels; these costs are assigned on a linear scale from 1 through 6.[3]
- *Mozilla exponential:* similar to Mozilla linear, but with fault severity values assigned on an exponential scale from 2^0 through 2^5.

linear level	exponential level	severity	percentage
1	1	trivial	2
2	2	minor	11
3	4	normal	6
4	8	major	76
5	16	critical	4
6	32	blocking	2

Table 3: Mozilla fault severity distributions.

Applying the unit fault severity distribution was trivial. Applying the two Mozilla fault severity distributions, however, was more difficult. The difficulty is that our prioritization techniques use information on module criticality in an attempt to prioritize in a manner that accounts for severity, but we did not have any historical data to support the estimation of module criticality. Thus, we required the generation of both module criticality and fault severity assignments.

Creating these two assignments independently cannot reflect any correlation between module criticalities and fault sever-

[2]For more information on Mozilla see www.mozilla.org. For more information on Mozilla testing and fault recording see bugzilla.mozilla.org.

[3]This distribution was obtained by querying the bugzilla database for "Resolved" bugs on the Netscape Browser for PC-windows2000.

ities, and the existence of such a correlation is a prerequisite for prioritization techniques that use module criticality to predict fault severity. Instead, our approach is to assume that a correlation between module criticalities and fault severities exists, and rely on this assumption in generating module criticality and fault severity assignments. To apply each fault severity distribution (other than unit) we generated a set of severity numbers with the required distribution, and randomly mapped those numbers to modules. We then considered each fault f, and assigned to f a severity number equal to the criticality number of the module containing f.

This approach does not allow us to investigate and compare prioritization techniques fairly, unless our investigations are restricted to conditional hypotheses beginning with the clause: "Suppose the assumption that a close correlation between module criticality and fault severity exists is valid". However, our focus in this study is not on the performance of prioritization techniques, but rather, on the effects that fault distributions and choices of severity values can have on APFD_C. In presenting our data, we take care to condition our conclusions to reflect our methodology.

Given these five test case cost distributions and three fault severity distributions, there are fifteen combinations of test case cost and fault distributions to consider. We restricted our attention to nine of these, as shown in Table 4.

	unit	random	normal	Mozilla	QTB
unit	X	X	X	X	X
Mozilla lin.	X			X	
Mozilla exp.	X			X	

Table 4: Fault severity distributions (left) versus test case cost distributions (above). Entries marked with "X" indicate combinations that were utilized in the study.

5 Results and Discussion

We present the results of our study in three steps. First, we analyze the impact of varying test case cost distributions across different prioritization techniques while maintaining a unit fault severity distribution. Second, we describe the effects of varying fault severity distributions across different prioritization techniques while maintaining a unit test case cost distribution. Third, we combine and analyze the effects of non-unitary distributions for both test case cost and fault severity. Interleaved with this presentation, we cite practical questions that might arise in applying test case prioritization, and discuss the answers to those questions.

Varying Test Case Cost Distributions

Figure 4 provides an initial view of the APFD_C values measured under different test case cost distributions in our study. The figure displays five sets of bars – one set presenting APFD_C values averaged across all runs with all techniques (left), and one set per technique presenting APFD_C values averaged across all runs with that technique. Each set of bars contains five individual bars – one for each test cost distribution studied. The height of each bar denotes the average APFD_C measured for test suites prioritized under the technique and distribution associated with that bar.[4]

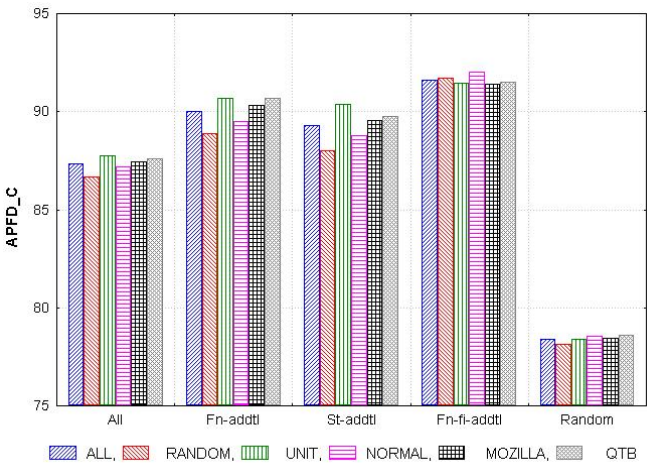


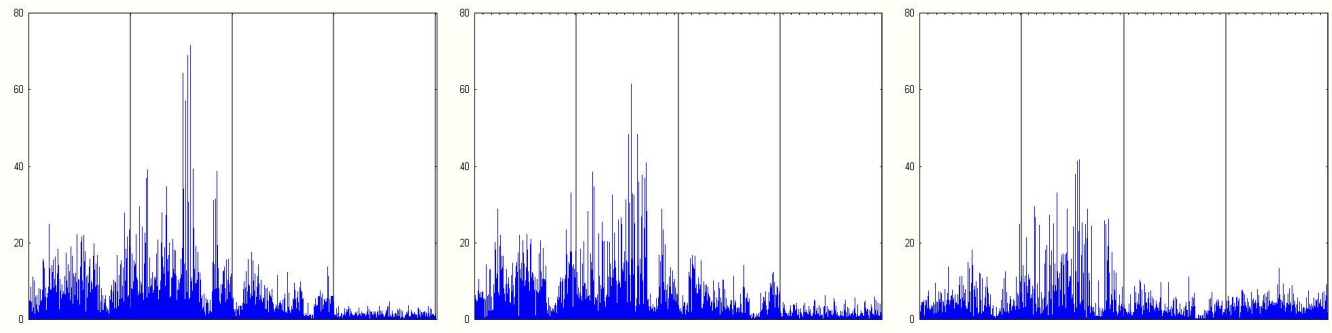
Figure 4: Mean APFD_C per distribution, per technique.

As the figure shows, in our study, test case cost distribution did have an impact on the overall average rate of fault detection of prioritized test suites as measured by APFD_C, across all runs with all techniques (as shown by the leftmost set of bars). These differences were statistically significant; however, they were not as large as we expected: the average overall APFD_C values for the different distributions differing by no more than 1 percentage point. Moreover, discriminating by technique (within each of the four rightmost sets of bars) we discovered that the extent of the impact varied with technique. For example, under st-addtl, the variance in average APFD_C across different distributions was statistically significant, whereas under fn-fi-addtl it was not.

The preceding analysis, however, was performed on average APFD_C values, and an investigation of individual differences in APFD_C paints a different picture. The graphs in Figure 5 illustrate the *absolute differences in APFD_C values* of prioritized test suites under the unit test case cost distribution compared to three of the four other distributions (subfigures A through C, respectively).[5] In each graph, the horizontal axis plots 2000 APFD_C observations: one for each of the 50 prioritized suites under each of the ten versions, for each of the four prioritization techniques. The observations are sorted by technique in order fn-addtl, st-addtl, fn-fi-addtl, random, and within technique by test suite and version. (The solid vertical lines in the figures delimit the boundaries between the data points for the four techniques.)

[4]Due to space limitations we do not include the complete statistical analysis of our data in this paper; however, it can be found in [6]. The complete analysis includes the results of applying ANOVA analysis and Tukey tests to all combinations of techniques and cost distributions.

[5]We omit the graph comparing the Unit distribution to the Mozilla distribution for space considerations; it is similar to the graph comparing the Unit distribution to the QTB distribution, and can be viewed in [6].



A. Absolute differences: Unit vs Random B. Absolute differences: Unit vs Normal C. Absolute differences: Unit vs QTB

Figure 5: Absolute differences in $APFD_C$ values across all observations, for three non-unit distributions vs. the unit distribution.

Figure 5 illustrates the extent to which, for individual prioritized suites, $APFD_C$ values under each distribution differ from $APFD_C$ values obtained with unit test case costs. In a few cases, the differences in $APFD_C$ exceed 50%, and in many cases they exceed 20%. This constitutes further evidence of the need to include test case cost distribution as an integral part of $APFD_C$ (failing to do so may provide poor prioritization). Further, because the unit test case cost distribution produces $APFD_C$ values equivalent to those produced by the (original) APFD metric, these differences show the extent to which the APFD and $APFD_C$ metrics differ.

Figure 5 also shows that under fn-addtl and st-addtl, test cost distributions exhibited higher variability in $APFD_C$ than under the other techniques (with st-addtl exhibiting the greatest variability). This suggests that for certain techniques the behavior of the distributions is more predictable than for others.

Finally, looking back at Figure 4 (previous page), all prioritization techniques provided (significant) improvements in $APFD_C$ over the random technique on average, under all test case cost distributions. Thus, even for a technique that is not the "best", and independently of test case cost distribution, prioritization improved rate of fault detection.

Practical Question 1. I am about to develop a new test suite for one of our products. What can I do to make the test suite more suitable for prioritization?

We conjecture that the use of many smaller test cases, rather than fewer larger ones, provides opportunities for test case scheduling, increasing the potential for prioritization gains. The Unit distribution with all small test case costs (and the Mozilla and QTB distributions to a lesser extent), for which average $APFD_C$ was higher than those of other distributions, provides initial evidence to support this conjecture. By partitioning larger test cases, the scope of the average test case decreases, allowing prioritization techniques to more precisely discriminate between test cases. Expensive test cases, even in small numbers, can limit the opportunities for prioritization (this is more likely when expensive test cases cover sections of the program that are likely to contain faults).

Varying Fault Severity Distributions

Our analysis of fault severity distributions show that fault severity distribution had a significant effect on the $APFD_C$ values for each prioritization technique. This is illustrated by Figure 6, which displays four sets of box plots (one set for each technique); each set of box plots contains one box plot for each of the three fault severity distributions, showing the distribution of $APFD_C$ values over the test suites prioritized by its associated technique and fault severity distribution.

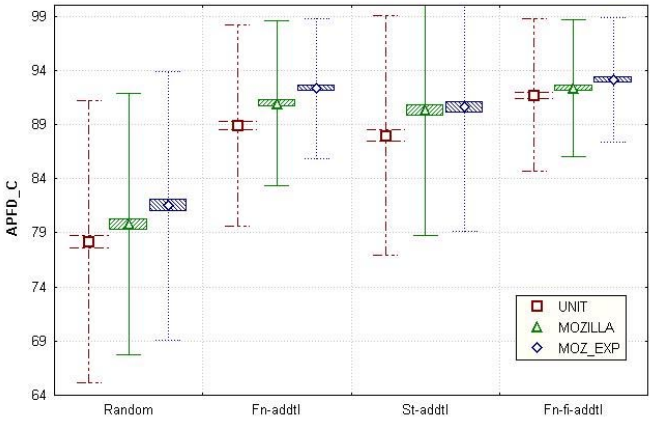


Figure 6: $APFD_C$ values, per distribution, per technique.

As the figure shows, fn-fi-addtl exhibited the most consistent behavior across distributions; it also exhibited better performance than other techniques. Random, in contrast, was the most susceptible to variation across distributions, and exhibited the poorest performance among the techniques.

Practical Question 2. I have access to two fault severity distributions A and B corresponding to previous releases of our product. I performed a small study using the two distributions, and found that distribution B provided a higher $APFD_C$ than distribution A on our previous release, but the difference in $APFD_C$s was not large. There is no difference in the costs of using the different distributions. How can I determine which distribution to use?

Even a small difference in $APFD_C$ can indicate that a particular distribution can find faults that may cause the most damage earlier in a testing cycle. Whether this difference will amount to a *practical* difference, however, depends on other factors. For example, if the test suite executes quickly it may make no sense to use prioritization in the first place. To answer the question, an illustration of the practical impact of the differences between $APFD_C$ values is needed.

One approach is to construct the graphs representing the $APFD_C$ scores for the two distributions on the previous release. Then, alter the horizontal axis label to reflect actual test execution costs (e.g. substituting time). From such a graph it will be more clear how the two orders compare.

To illustrate, we performed such an analysis on one version of `space` under the fn-fi-addtl technique. In this case, the $APFD_C$ for unit test case cost with unit fault severity was 80%, the $APFD_C$ for unit test case cost with Mozilla linear fault severity was 84%, and the APFD for unit test case cost with Mozilla exponential fault severity was 93%.

The graph in Figure 7 depicts the tops of the $APFD_C$ curves for these 3 distributions, with the x-axis representing the percentage of total test case costs incurred and the y-axis representing the percentage of total fault severity detected. After having incurred only 2% of the test case costs, the differences between distributions has become evident. The Mozilla-exponential fault severity distribution has captured 80% of the accumulated fault severity, the Mozilla-linear fault severity distribution has captured approximately 50%, and the unit fault severity distribution has captured less than 35%. The disparity among distributions grows smaller as additional test cost is incurred, and by the time 40% of the test cost has been accounted for, 100% of the fault severity has been captured by all three distributions.

What is the practical impact of these differences? Suppose first that the scale below the x-axis represents the time in days required for the test suite to be executed. In this case, under the Mozilla-exponential distribution, 80% of the total fault severity has been accounted for within 4 days, whereas under the Mozilla-linear distribution the same severity is not accounted for until after approximately 19 days. These savings may be significant if the product must be shipped before the testing process is completed, because the most severe faults are more likely to have been detected.

Suppose, however, that the execution of the test suite is accomplished over the weekend and its cost can be measured in hours instead of days. In this case, the differences between the distributions may not be practically significant. Thus, as the overall testing costs diminish, the impact of choosing a particular distribution is lessened.

Varying Test Cost and Fault Severity Distributions
Having analyzed the effects of varying test case cost and fault severity distributions individually, we now consider the

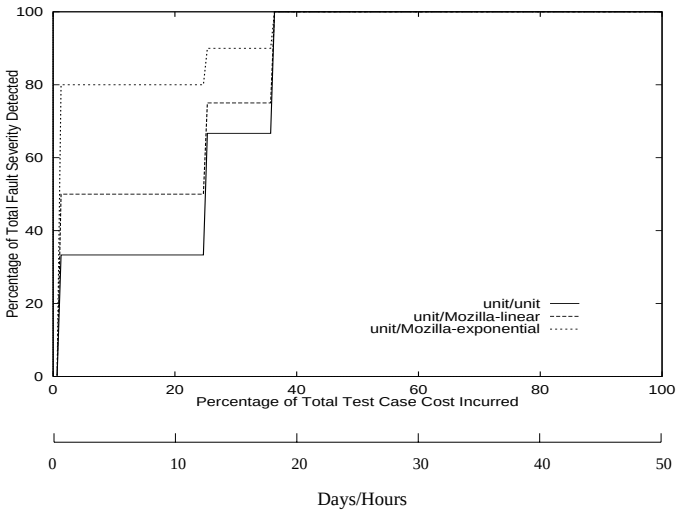


Figure 7: $APFD_C$ curves for Practical Question 2.

results obtained by varying both distributions concurrently. To simplify the presentation, we focus on the behavior of $APFD_C$ under just 50 randomly sampled cases of st-addtl.

Figure 8 presents a scatter plot to represent three combined distributions: (1) unit test case cost with unit fault severity, (2) Mozilla test case cost with linear Mozilla fault severity (3) Mozilla test case cost with exponential Mozilla fault severity. The x-axis represents the $APFD_C$ value under the unit-unit distribution, and each point plotted represents the value of $APFD_C$ under one of the other two distributions.

The plot shows that the $APFD_C$ values for the unit-unit distribution were significantly different from the $APFD_C$ values under the other distributions; this is evident from the large variance in the plots for both Mozilla distributions. The choice of different combinations of test case cost and fault severity distributions did thus have an impact on $APFD_C$.

To further illustrate the differences among the distributions we fitted the individual cases with linear equations using regression analysis, and drew 95% confidences intervals around the regression lines. In the figure, the central line for each distribution is the regression line for that distribution; the other two lines (bands) for each distribution represent the probability that the "true" fitted line (within the population of points considered) falls between the bands. The lines suggest that the Mozilla/Mozilla-exponential distribution combination provides the highest $APFD_C$ values, but as $APFD_C$ becomes closer to 100, both of the distribution combinations involving Mozilla converge.

Practical Question 3. What considerations should I attend to when specifying test cost and fault severity distributions?

Applying linear transformations to a scale of test costs or fault severities does not affect $APFD_C$ values. For example, given a three-level fault severity ranking scheme, choosing severity values 1, 2, and 3 is equivalent to choosing 10, 20

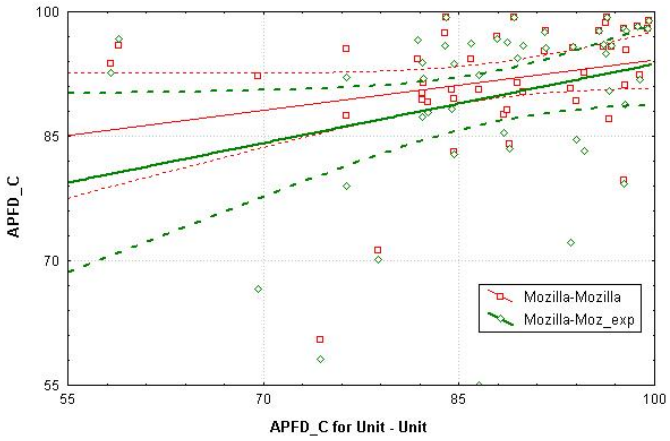


Figure 8: APFD$_C$ values for combined distributions.

and 30 where evaluation of test case orders is concerned, because it does not affect the units-of-fault-severity-detected-per-unit-test-cost-values of test cases relative to one another. In contrast, applying non-linear transformations to a scale of test case costs or fault severities can affect APFD$_C$ values because it can alter the relative worth of test case orders as shown by the Mozilla-exp distribution presented above.

6 CONCLUSIONS

We have presented a new metric, APFD$_C$, for assessing the rate of fault detection of prioritized test cases, that incorporates varying test case and fault costs. We have also presented techniques for prioritizing test cases that attempt to account for the effects of varying test case cost and fault severity. Our case study applying these techniques illustrates the application of the metric and several effects that arise in the use of different test cost and fault severity distributions. The results of the study also highlight differences between the previous and new metrics.

Our focus in this work has been the APFD$_C$ metric. We have presented a few prioritization techniques intended to improve prioritization under this metric, but there may be other techniques better suited to maximizing the objective function represented by APFD$_C$. Furthermore, practitioners currently do prioritize test cases by other mechanisms not investigated here. Experiments with such other approaches would be enlightening.

We are currently constructing infrastructure with which to experiment further with our metrics and techniques; such infrastructure will support both rigorous experiments in which factors such as test cost and fault severity can be controlled and varied, and case studies using real systems, with test cost and fault severity data drawn directly from those systems. In addition to providing further opportunities to evaluate the metric and techniques, such experiments will provide data with which we can create more rigorous guidelines and assessment tools for evaluating techniques and distributions. With such guidelines and tools, we hope to help practitioners answer the practical questions they face in deploying test case prioritization.

ACKNOWLEDGEMENTS

This work was supported in part by the NSF Information Technology Research program under Awards CCR-0080898 and CCR-0080900 to University of Nebraska, Lincoln and Oregon State University, The work was also supported in part by NSF Awards CCR-9703108 and CCR-9707792 to Oregon State University, and also by a NASA-Epscor Space Grant Award to the University of Nebraska, Lincoln. Alberto Pasquini, Phyllis Frankl, and Filip Vokolos shared the `space` program and test cases. Roland Untch, Mary Jean Harrold, and Chengyun Chu contributed to earlier stages of the work.

REFERENCES

[1] L. C. Briand, J. Wust, S. Ikonomovski, and H. Lounis. Investigating quality factors in object oriented designs: an industrial case study. In *Proc. Int'l. Conf. on Softw. Eng.*, pages 345–354, May 1999.

[2] S. Elbaum, A. Malishevsky, and G. Rothermel. Prioritizing test cases for regression testing. In *Proc. Int'l. Symp. Softw. Testing and Analysis*, pages 102–112, Aug. 2000.

[3] S. G. Elbaum and J. C. Munson. Code churn: A measure for estimating the impact of code change. In *Proc. Int'l. Conf. Softw. Maint.*, pages 24–31, Nov. 1998.

[4] S. G. Elbaum and J. C. Munson. Software evolution and the code fault introduction process. *Emp. Softw. Eng. J.*, 4(3):241–262, Sept. 1999.

[5] F. Lanubile, A. Lonigro, and G. Visaggio. Comparing models for identifying fault-prone software components. In *Proc. 7th Int'l. Conf. on Softw. Engr. and Knowledge Engr.*, pages 12–19, June 1995.

[6] A. G. Malishevsky, S. Elbaum, and G. Rothermel. Incorporating varying test costs and fault severities into test case prioritization. Technical Report 00-60-09, Oregon State University, Aug. 2000.

[7] J. C. Munson. Software measurement: Problems and practice. *Annals of Softw. Eng.*, 1(1):255–285, 1995.

[8] J. Musa. *Software Reliability Engineering*. McGraw-Hill, New York, NY, 1999.

[9] A. P. Nikora and J. C. Munson. Software evolution and the fault process. In *Proc. Twenty Third Annual Softw. Eng. Workshop, NASA/Goddard Space Flight Center*, 1998.

[10] K. Onoma, W.-T. Tsai, M. Poonawala, and H. Suganuma. Regression testing in an industrial environment. *Comm. ACM*, 41(5):81–86, May 1988.

[11] G. Rothermel, R. Untch, C. Chu, and M. J. Harrold. Test case prioritization: an empirical study. In *Proc. Int'l. Conf. Softw. Maint.*, pages 179–188, Aug. 1999.

[12] G. Schulmeyer and J. McManus. *Handbook of Software Quality Assurance*. Prentice Hall, New York, NY, 3rd edition, 1999.

[13] W. Wong, J. Horgan, S. London, and H. Agrawal. A study of effective regression testing in practice. In *Proc. Eighth Intl. Symp. on Softw. Rel. Engr.*, pages 230–238, Nov. 1997.