

Modeling the Cost-Benefits Tradeoffs for Regression Testing Techniques

Alexey G. Malishevsky and Gregg Rothermel
Computer Science Department
Oregon State University
Corvallis, OR
{malishal,grother}@cs.orst.edu

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

Abstract

Regression testing is an expensive activity that can account for a large proportion of the software maintenance budget. Because engineers add tests into test suites as software evolves, over time, increased test suite size makes revalidation of the software more expensive. Regression test selection, test suite reduction, and test case prioritization techniques can help with this, by reducing the number of regression tests that must be run and by helping testers meet testing objectives more quickly. These techniques, however, can be expensive to employ and may not reduce overall regression testing costs. Thus, practitioners and researchers could benefit from cost models that would help them assess the cost-benefits of techniques. Cost models have been proposed for this purpose, but some of these models omit important factors, and others cannot truly evaluate cost-effectiveness. In this paper, we present new cost-benefits models for regression test selection, test suite reduction, and test case prioritization, that capture previously omitted factors, and support cost-benefits analyses where they were not supported before. We present the results of an empirical study assessing these models.

1 Introduction

Software maintenance can account for a large proportion of the overall cost of software [11, 17]. Regression testing is an important part of maintenance, performed when software is modified to help ensure that it has not lost required functionality. Because regression testing is performed frequently, it can account for a large proportion of maintenance costs [1, 10, 17].

To reduce the cost of regression testing, engineers often save their test suites for use in retesting that software. As the software evolves, they add tests to these suites. As a result, the costs of maintaining and using the test suites increase.

Several techniques have been devised for dealing with these costs. *Regression test selection techniques* select a subset of the existing test suite for execution, depending on factors such as the changes made to the code and the exe-

cutation behavior of tests. *Test suite reduction techniques*¹ permanently reduce the test suite by identifying and discarding redundant tests. *Test case prioritization techniques* retain the entire test suite, but order the tests prior to execution, attempting to help testing objectives be met earlier during testing. (Henceforth, for brevity, we shall usually refer to these techniques simply as “selection,” “reduction,” and “prioritization,” respectively.)

The foregoing techniques can be beneficial to testers. Experiments [9, 15, 19, 21] show that reduction can be very effective in reducing test suite size. Other experiments [2, 8, 9, 15, 17] show that selection can effectively reduce the number of tests that must be rerun. Still other experiments [5, 6, 20] show that prioritization can allow test suites to detect faults earlier in testing than would otherwise be possible, allowing engineers to address those faults earlier.

Such evidence shows that selection, reduction, and prioritization can produce savings relative to the common practice of rerunning all tests without regard to test order. However, the savings demonstrated do not guarantee the cost-effectiveness of the techniques, because the techniques also have associated costs. For example, selection techniques have analysis costs [11]. Also, reduced test suites could miss some faults which would otherwise be detected by the original test suite [15, 19, 21]. Finally, detecting faults earlier may matter only when the ratio of fault costs to test costs is sufficiently high [6].

Practitioners wishing to use regression testing techniques, and researchers wishing to study those techniques, need methods with which to assess the cost-effectiveness of those techniques. To support such assessments, cost-benefits models are required which take into account the factors affecting the costs and benefits of the techniques.

Leung and White [11] present such a model for regression test selection. They consider several constituent costs of selection techniques (including analysis, test execution, validation, and selection costs) and use their model to provide a cost-benefits comparison of test selection techniques

¹Also referred to as “test suite minimization” techniques [9, 21].

to the retest-all approach (rerunning all tests). However, this model does not consider the cost of missing faults due to discarding tests, and can show that a technique is cost-effective even though it lets detectable faults through.

Wong et al. [21] experiment with test suite reduction and study its costs and benefits. To assess their results the authors measure reduction in test suite size (benefits) and reduction in the number of faults detectable by a test suite (cost). These measures imply an interest in both of these factors; however, the authors do not explicitly present a cost model, and do not consider factors such as analysis costs.

Previous work on prioritization [5, 6, 20] provides a metric with which to compare prioritization techniques in terms of rate of fault detection. However, this metric does not support assessments, in terms of resources such as time or money, of the relative cost benefits of techniques, or of whether a given technique is cost-effective.

Thus, although prior work makes progress in assessing the merits of regression testing techniques, existing models omit important factors or are inappropriate for use in quantitatively assessing the relative cost benefits of various regression testing techniques.

This paper presents new models for assessing the relative cost-benefits of selection, reduction, and prioritization techniques. These models allow us to compare different techniques, determine when techniques would be beneficial to employ, and determine which techniques are better in given situations than others. We present the results of experiments that show the usefulness of the models in assessments.

2 Background and Related Work

2.1 Regression Testing and Techniques

Let P be a program, P' be a modified version of P , and T be a test suite developed for P . Regression testing seeks to test P' . To facilitate such testing, test engineers may reuse T to the extent possible, but new test cases may also be required. Both reuse of T and creation of new test cases are important; however, it is test reuse that concerns us here.

When P is modified, creating P' , test engineers may simply reuse all non-obsolete² test cases in T to test P' ; this is known as the *retest-all* technique [11].

The *retest-all* technique can be expensive: rerunning all test cases may require an unacceptable amount of time or human effort. Regression test selection (selection) techniques [4, 12, 17] use information about P , P' , and T to select a subset of T with which to test P' . (For a survey of selection techniques, see [16].)

As P evolves, new test cases may be added to T to validate new functionality. Over time, T grows, and its test cases become redundant in terms of code or functionality exercised. Test suite reduction (reduction) techniques [3, 9, 13] address this problem by using information

²Test cases in T that no longer apply to P' are *obsolete*, and must be reformulated or discarded [11].

about P and T to permanently remove redundant test cases from T , so that subsequent reuse of T can be more efficient. Reduction thus differs from selection in that the latter does not permanently remove test cases from T , but simply “screens” those test cases for use on a specific version P' of P , retaining unused test cases for use later.

Test case prioritization (prioritization) techniques [5, 22] schedule test cases so that those with the highest priority, according to some criterion, are executed earlier in the regression testing process than lower priority test cases. For example, testers might wish to schedule test cases in an order that achieves code coverage at the fastest rate possible, exercises features in order of expected frequency of use, or increases the likelihood that those test cases will detect faults early in testing.

2.2 Previous Cost Models

There have been no explicit cost models presented for reduction; however, the research literature contains cost models for both selection and prioritization.

Leung and White present a cost model for regression test selection in [11]. Their model incorporates various costs of regression testing, including the costs of executing and validating test cases and the cost of performing analyses to support test selection, and provides a way to compare techniques for relative effectiveness. Applied to *safe* regression test selection techniques [17], which necessarily select (under certain conditions) all test cases in the existing test suite that may reveal faults, this model is appropriate. However, the model does not consider the cost of missing faults due to discarding tests. Thus, applied to techniques that are not safe, the model can show that a given technique is cost-effective due to savings in test execution costs even though that technique allows faults to go undetected.

To allow comparisons of prioritization techniques, in previous work [5] we presented a metric, $APFD_C$, that measures the weighted average of the percentage of faults detected over the execution of a test suite. $APFD_C$ values range from 0 to 100; higher numbers imply faster fault detection rates. More formally, let T be a test suite containing n test cases, let F be a set of m faults revealed by T , let f_i be the cost of the i -th fault, and let e_j be the cost of the j -th test. Let TF_i be the first test case in order O of T that reveals fault i . The $APFD_C$ for T is given by the equation:

$$APFD_C = \frac{\sum_{i=1}^m (f_i \times (\sum_{j=TF_i}^n e_j - \frac{1}{2} e_{TF_i}))}{\sum_{i=1}^m e_i \times \sum_{i=1}^m f_i} \quad (1)$$

This measure has proven successful for comparing different prioritization techniques; however, it does not support assessments, in terms of resources such as time or money, of the relative cost benefits of techniques, or whether a given technique is cost-effective. Thus, the method may not easily support decisions about when to use various techniques.

3 Cost Models

We now present our cost models for selection, reduction, and prioritization. We consider costs associated with each regression testing methodology when a given technique is used and under common practice; we then compare those costs, and perform some simplifications of formulas.

Note that the models we present are intended to be evaluative, not predictive. As such they could be used by practitioners on historical data to investigate which techniques might have been most cost-effective for their systems and testing processes, and used to guide future efforts. They can also be used by researchers when experimenting with techniques, in order to evaluate them.

In the analyses, let P be a given program, let P' be a modified version of P , let T be the test suite for P , and consider the application of the regression testing technique relative to P and P' . Also, let $F(T)$ be a set of regression faults in P detected by test suite T . Because we consider P to be constant, all factors depending on P will be constant relative to T . In other words, the program is treated as a constant, not as a variable.

3.1 Regression Test Selection

Our model for selection builds on Leung and White's [11] model outlined in Section 2. Let T' be the selected test suite. We define the following variables:

- $Ca(T)$ - cost of analysis.
- $Ce(T)$ - cost of execution.
- $Cc(T)$ - cost of result checking.
- $Cs(T)$ - cost of selection.
- $Cm(T)$ - cost of maintenance of the test suite.
- $Cf(F(T) \setminus F(T'))$ - cost of omitting faults by not selecting $T \setminus T'$.

$Ca(T)$ includes the cost of source code analysis, analysis of changes between old and new versions, and collection of execution traces. $Ce(T)$ includes the cost of preparing the program for execution and running tests from T . $Cc(T)$ includes the cost of validation of program outputs by either automatic comparison with previously saved results or manual inspection of outputs. $Cs(T)$ includes the cost of running a test selection tool. $Cf(F(T) \setminus F(T'))$ includes the cost of missing faults such as degraded customer satisfaction and confidence in the software, litigation, or the cost of releasing updates to fix faults. $Cm(T)$ includes the cost of maintaining the database of tests and program outputs from previous executions. Some of these costs can be related to time (later release date), but all costs can be converted to monetary values such as engineers' salaries, equipment rental, and so on.

The cost of the retest-all strategy (where all tests are executed) is

$$C = Ce(T) + Cc(T) + Cm(T) \quad (2)$$

The cost of a regression test selection strategy (where only part of a test suite is executed) is

$$C' = Ca(T) + Ce(T') + Cc(T') + Cs(T) + Cf(F(T) \setminus F(T')) + Cm(T) \quad (3)$$

(the whole test suite must still be maintained for future applications of regression testing).

Selection is more cost effective than retest-all iff $C' < C$; in which case:

$$Ca(T) + Ce(T') + Cc(T') + Cs(T) + Cf(F(T) \setminus F(T')) + Cm(T) < Ce(T) + Cc(T) + Cm(T) \quad (4)$$

This simplifies to:

$$Ca(T) + Ce(T') + Cc(T') + Cs(T) + Cf(F(T) \setminus F(T')) < Ce(T) + Cc(T) \quad (5)$$

We now make several assumptions to simplify this inequality. We assume that $Ca(T) = A + a|T|$, where A is the fixed, not test dependent, cost of analysis, and a is the cost of additional analysis per test. We also assume that $Ce(T) = E + e|T|$, where E is the fixed, not test dependent, cost of running T (startup and clean up costs), and e is the additional cost per test. Finally, we assume that $Cc(T) = c|T|$, where c is the result validation cost per test.

These assumptions are reasonable because analysis time includes some non-test-dependent costs related to program analysis, such as representation and instrumentation, and some test-related costs related to activities such as trace collection. Also, execution time has some startup cost and runtime for each test suite, and then individual costs per test. (These individual costs vary per test; however, to simplify the situation, we shall assume that test costs are the same.) Result checking time is directly proportional to the number of tests (of course, result checking times also differ per test; however, we shall assume uniform costs).

With these assumptions, inequality (5) becomes:

$$A + a|T| + E + e|T'| + c|T'| + Cs(T) + Cf(F(T) \setminus F(T')) < E + e|T| + c|T| \quad (6)$$

Simplifying this, we have:

$$A + a|T| + e|T'| + c|T'| + Cs(T) + Cf(F(T) \setminus F(T')) < e|T| + c|T| \quad (7)$$

In our own experience $A \ll a|T|$ [2, 17], so we can rewrite inequality (7) as follows:

$$a|T| + e|T'| + c|T'| + Cs(T) + Cf(F(T) \setminus F(T')) < e|T| + c|T| \quad (8)$$

Also, we partition the original suite T into two subsets: T_o (tests that are not selected) and T_n (tests that are selected). In other words, $T_n = T'$ and $T_o = T \setminus T'$. This yields:

$$a|T_o| + a|T_n| + e|T_n| + c|T_n| + Cs(T_o \cup T_n) + Cf(F(T_o \cup T_n) \setminus F(T_n)) < e|T_o| + e|T_n| + c|T_o| + c|T_n| \quad (9)$$

which simplifies to:

$$a|T_o| + a|T_n| + Cs(T_o \cup T_n) + Cf(F(T_o \cup T_n) \setminus F(T_n)) < e|T_o| + c|T_o| \quad (10)$$

Test selection cost has two components: non-test dependent cost and test dependent cost. To simplify the analysis we assume that its relationship with the number of tests is linear: $Cs(T) = S + s|T|$ with S representing the non-test dependent cost and s representing the additional cost per test. In this case, inequality (10) becomes:

$$a|T_o| + a|T_n| + S + s|T_o| + s|T_n| + Cf(F(T_o \cup T_n) \setminus F(T_n)) < e|T_o| + c|T_o| \quad (11)$$

However, in our experience S is small relative to other costs, so we can ignore it.

Another simplifying assumption involves the severity of faults: we assume that fault severity is uniform. This makes the cost of missing faults directly proportional to the number of missed faults (assuming the fact that faults are missed does not add additional cost), thus $Cf(x) = f|x|$, where f is the cost of omitting one fault and x is some set of faults. This yields:

$$a|T_o| + a|T_n| + s|T_o| + s|T_n| + f|F(T_o \cup T_n) \setminus F(T_n)| < e|T_o| + c|T_o| \quad (12)$$

Finally, let $|T_o| = \alpha n$, where $1 - \alpha$ is the proportion of a test suite selected, then $|T_n| = (1 - \alpha)n$, where $n = |T|$. This yields:

$$an + sn + f|F(T_o \cup T_n) \setminus F(T_n)| < e\alpha n + c\alpha n \quad (13)$$

which simplifies to

$$(a + s)n + f|F(T_o \cup T_n) \setminus F(T_n)| < (e + c)\alpha n \quad (14)$$

This model expresses the relationship between cost factors, the number of selected tests, and the number of missed faults. Engineers can use these inequalities to assess whether selection is cost beneficial. If, after substituting variables, the inequality holds, selection is cost effective, otherwise, it is not. A measure of cost savings associated with selection can also be calculated by subtracting the left hand side from the right hand side.

We can also use this model to compare the cost-effectiveness of two selection techniques. Let T' be the test suite selected by technique 1 and let T'' be the test suite selected by technique 2. Regression testing costs are

$$C_1 = Ca_1(T) + Ce(T') + Cc(T') + Cs_1(T) + Cf(F(T) \setminus F(T')) \quad (15)$$

and

$$C_2 = Ca_2(T) + Ce(T'') + Cc(T'') + Cs_2(T) + Cf(F(T) \setminus F(T'')) \quad (16)$$

when techniques 1 and 2 are employed, respectively. (Analysis costs can differ across techniques because they may need different data, and selection costs can differ because different techniques may use different selection algorithms.) The technique with the lowest cost C_k is the most cost-beneficial.

3.2 Reduction

When constructing the cost model for selection, we were able to work from Leung and White's [11] model. There are no comparable models for reduction; however, a similar strategy to constructing one applies, as follows.

There are important differences between selection and reduction. In selection, for each version v , a subset of T is selected, executed, and validated on v . On the other hand, in reduction, following release of version v , a subset T' of T is selected while other tests are permanently discarded, and T' is then executed and validated on subsequent versions as they are released. In our cost model for reduction, we assume that after test suite T is reduced, regression testing will be conducted g times.

We use the variables $Ca(T)$, $Ce(T)$, $Cc(T)$, $Cs(T)$, and $Cm(T)$, defined in the preceding section, corresponding to the same factors. We also define the following variable $Cf(F_k(T) \setminus F_k(T'))$ to be the cost of omitting faults, where $F_k(x)$ is the set of regression faults on version v_k detected by a test suite x .

The cost of the retest-all strategy (where all tests are executed) is:

$$C = g \times Ce(T) + g \times Cc(T) + g \times Cm(T) \quad (17)$$

The cost of a reduction strategy (where part of a test suite is discarded) is:

$$C' = Ca(T) + g \times Ce(T') + g \times Cc(T') + Cs(T) + \sum_{1 \leq k \leq g} Cf(F_k(T) \setminus F_k(T')) + g \times Cm(T') \quad (18)$$

Reduction is more cost-effective than retest-all iff $C' < C$, in which case:

$$Ca(T) + g \times Ce(T') + g \times Cc(T') + Cs(T) + \sum_{1 \leq k \leq g} Cf(F_k(T) \setminus F_k(T')) + g \times Cm(T') < g \times Ce(T) + g \times Cc(T) + g \times Cm(T) \quad (19)$$

Following reasoning similar to that used for selection, we now make several assumptions to simplify this inequality. We assume that $Ca(T) = A + a|T|$, where A is the fixed, not test dependent, cost of the analysis, and a is the cost of additional analysis per test. We assume that $Ce(T) = E + e|T|$, where E is the fixed, not test dependent, cost of running T (startup and clean up costs), and e is the additional cost per test. We also assume that $Cc(T) = c|T|$, where c is the result validation cost per test. Finally, we assume that $Cm(T) = M + m|T|$, where M is the non-test dependent cost of maintenance and m is the additional maintenance cost per test.

With these assumptions, inequality (19) becomes:

$$A + a|T| + g \times (E + e|T'|) + g \times c|T'| + Cs(T) + \sum_{1 \leq k \leq g} Cf(F_k(T) \setminus F_k(T')) + g \times (M + m|T'|) < g \times (E + e|T|) + g \times c|T| + g \times (M + m|T|) \quad (20)$$

Simplifying this, we have:

$$\begin{aligned}
& A + a|T| + g \times e|T'| + g \times c|T'| + Cs(T) \\
& + \sum_{1 \leq k \leq g} Cf(F_k(T) \setminus F_k(T')) + g \times m|T'| \\
& < g \times e|T| + g \times c|T| + g \times m|T| \quad (21)
\end{aligned}$$

In our experience $A \ll a|T|$ [2, 17], so we rewrite (21) as:

$$\begin{aligned}
& a|T| + g \times e|T'| + g \times c|T'| + Cs(T) \\
& + \sum_{1 \leq k \leq g} Cf(F_k(T) \setminus F_k(T')) + g \times m|T'| \\
& < g \times e|T| + g \times c|T| + g \times m|T| \quad (22)
\end{aligned}$$

Also, we partition the original test suite T into two subsets: T_o (tests which are discarded) and T_n (tests which are kept). In other words, $T_n = T'$ and $T_o = T \setminus T'$. This yields:

$$\begin{aligned}
& a|T_o| + a|T_n| + g \times e|T_n| + g \times c|T_n| \\
& + Cs(T_o \cup T_n) + \sum_{1 \leq k \leq g} Cf(F_k(T_o \cup T_n) \setminus F_k(T_n)) \\
& + g \times m|T_n| < g \times e|T_o| + g \times e|T_n| + g \times c|T_o| \\
& + g \times c|T_n| + g \times m(|T_n| + |T_o|) \quad (23)
\end{aligned}$$

which simplifies to:

$$\begin{aligned}
& a|T_o| + a|T_n| + Cs(T_o \cup T_n) + \sum_{1 \leq k \leq g} Cf(F_k(T_o \cup T_n) \setminus F_k(T_n)) \\
& < g \times e|T_o| + g \times c|T_o| + g \times m|T_o| \quad (24)
\end{aligned}$$

If we perform regression testing enough times, such that the cost of testing is much higher than analysis and selection costs, we can neglect those costs. The inequality becomes:

$$\begin{aligned}
& \sum_{1 \leq k \leq g} Cf(F_k(T_o \cup T_n) \setminus F_k(T_n)) \\
& < g \times (e|T_o| + c|T_o| + m|T_o|) \quad (25)
\end{aligned}$$

We again assume that fault severity is uniform. This makes the cost of missing faults directly proportional to the number of missed faults (assuming the fact that faults are missed does not add additional cost). This yields $Cf(x) = f|x|$, where f is the cost of omitting one fault and x is some set of faults. This yields:

$$\begin{aligned}
& \sum_{1 \leq k \leq g} (f|F_k(T_o \cup T_n) \setminus F_k(T_n)|) \\
& < g \times (e|T_o| + c|T_o| + m|T_o|) \quad (26)
\end{aligned}$$

Finally, letting $|T_o| = \alpha n$, then $|T_n| = (1 - \alpha)n$, where $n = |T|$ and α is the proportion of the test suite discarded. This yields:

$$\begin{aligned}
& \sum_{1 \leq k \leq g} (f|F_k(T_o \cup T_n) \setminus F_k(T_n)|) \\
& < g \times (e\alpha n + c\alpha n + m\alpha n) \quad (27)
\end{aligned}$$

and finally,

$$\sum_{1 \leq k \leq g} (f|F_k(T_o \cup T_n) \setminus F_k(T_n)|) < g(e + c + m)\alpha n \quad (28)$$

This model expresses the relationship between cost factors, the number of selected tests, and the number of missed faults. As with the model for selection, engineers can

use these inequalities to assess whether reduction is cost-effective by substituting variables and determining whether the inequality holds. A measure of cost savings associated with reduction can be calculated by subtracting the left hand side from the right.

We can use this model to compare reduction techniques in terms of cost-effectiveness. Let T' be the test suite produced by technique 1 and let T'' be the test suite produced by technique 2. Regression testing costs are

$$\begin{aligned}
C_1 &= Ca_1(T) + g \times Ce(T') + g \times Cc(T') + Cs_1(T) \\
& + \sum_{1 \leq k \leq g} Cf(F_k(T) \setminus F_k(T')) + g \times Cm(T') \quad (29)
\end{aligned}$$

and

$$\begin{aligned}
C_2 &= Ca_2(T) + g \times Ce(T'') + g \times Cc(T'') + Cs_2(T) \\
& + \sum_{1 \leq k \leq g} Cf(F_k(T) \setminus F_k(T'')) + g \times Cm(T'') \quad (30)
\end{aligned}$$

when the first and second techniques are employed, respectively. As with selection, analysis and reduction costs may differ between techniques. The technique which has the lowest cost C_k will be the most cost-beneficial.

3.3 Prioritization

Finally, we present a model to support cost-benefits analyses of prioritization techniques. Here, as described in Section 2, a previous model exists, but that model does not support cost-benefits analyses. In this case however, the model differs substantially from the model for selection.

Given a test suite T , we define the following variables:

$Ca(T)$ is the cost of analysis.

$Cp(T)$ is the cost of the prioritization algorithm.

In discussing these variables, we distinguish two phases of regression testing — the preliminary and critical phases — these being the times before and after the release is available for testing. Preliminary phase activities may be assigned different costs than critical phase activities, since the latter may have greater ramifications for things like release time.

$Ca(T)$ is as described in Section 3.1, and can be performed during the preliminary phase of testing. $Cp(T)$ is the actual cost of running a prioritization tool, and, depending on the prioritization algorithm used, can be performed during either the preliminary or critical phase.

Given an order O of test suite T , suppose a fault f is revealed by the k th test occurring after $d_f^O = \sum_{i=1}^k e_i^O$ time units, where e_i^O is the execution and validation time of test i in the test suite under order O . Suppose that we have two orders O' and O'' of the same test suite T . Suppose fault f is revealed at time $d_f^{O'}$ if the test suite T is under order O' and the same fault is revealed at time $d_f^{O''}$ if the test suite T is under order O'' . Suppose that $d_f^{O'} < d_f^{O''}$ (f is revealed earlier under order O'). If we have just this one fault, ordering O' is beneficial relative to ordering O'' ,

because it saves $d_f^{O''} - d_f^{O'}$ units of time.

Consider a more complicated case in which T detects m faults. Let TF_i^O to be the test number under order O which first detects fault i . Let the test suite contain n tests.

Define:

$$r_{ik}^O = \begin{cases} 1 & \text{if test } k \text{ in the test suite } T \\ & \text{under order } O \text{ reveals fault } i \\ 0 & \text{otherwise} \end{cases} \quad (31)$$

r_{ik}^O shows whether test k under order O reveals fault i .

Let x_i^O be the set of indices of tests under order O which reveal fault i . Formally:

$$x_i^O = \{k \mid \forall 1 \leq k \leq n \ r_{ik}^O = 1\} \quad (32)$$

Given these definitions, $TF_i^O = \min(x_i^O)$ (the minimum of all elements in set x_i^O).

We define $delays^O$ as the cumulative cost of waiting for each fault to be exposed while executing test suite T under order O :

$$delays^O = \sum_{i=1}^m \left(\left(\sum_{k=1}^{TF_i^O} e_k^O \right) \times f_i \right) \quad (33)$$

In equation (33), m is the number of faults, e_k^O is the runtime and validation time of test k in suite T under order O , and f_i is the cost of waiting a time unit for a fault i to be exposed (e.g., paying a programmer to wait for a given fault to be exposed in order to attempt to correct it, where different faults require different programmers with different salaries). Essentially, when $\forall i \ f_i = 1$, $delays^O$ sums, for each fault, the time between the start of test suite execution and the time when this fault is first revealed.

Given that the cost savings due to the application of a prioritization technique that creates order O'' relative to using a random order³ O' is

$$delays^{O'} - delays^{O''} - Ca(T) - Cp(T), \quad (34)$$

we can define the cost of random ordering as

$$C^{O'} = delays^{O'} \quad (35)$$

and the cost of prioritized order O'' as

$$C^{O''} = delays^{O''} + Ca(T) + Cp(T) \quad (36)$$

Technically, formulas for $C^{O'}$ and $C^{O''}$ do not describe physical costs, because the costs associated with $delays^O$ may not materialize in practice - they depend on factors such as availability of personnel. However, their difference shows us an upper bound on possible cost savings which could be achieved given favorable conditions. In other words, earlier detection of faults can lead to more efficient use of human resources and earlier release dates.

As stated previously, prioritization is cost-effective iff:

$$delays^{O''} + Ca(T) + Cp(T) < delays^{O'} \quad (37)$$

³Because there are many random orderings, this cost can be defined as the average cost over all possible random orderings of test suite T . However, because it is not practical to calculate this cost in this manner, we estimate this cost by computing an average for a fixed number of randomly chosen orderings.

We now make several assumptions to simplify this inequality. We assume that $Ca(T) = A + a|T|$, where A is the non-test dependent cost and a is the additional cost per test. Prioritization cost is usually linear or quadratic in some measure of the size of the test suite, depending on the algorithm [6]. However, in our experience, the cost of performing prioritization is small compared to other costs, and because it is machine time, it can be neglected.

With these assumptions, inequality (37) becomes:

$$delays^{O''} + A + a|T| < delays^{O'} \quad (38)$$

Also, in our experience, $A \ll a|T|$ [2, 18], so we can rewrite inequality (38) as follows

$$delays^{O''} + a|T| < delays^{O'} \quad (39)$$

This model lets us compare prioritization techniques and answer the question whether a given prioritization technique will be cost-beneficial compared to random ordering.

If we need to compare two prioritization techniques 1 and 2 which produce orders O_1 and O_2 , respectively, we will have to compute $C_1 = Ca_1(T) + Cp_1(T) + delays^{O_1}$ and $C_2 = Ca_2(T) + Cp_2(T) + delays^{O_2}$ for techniques 1 and 2, respectively (analysis and prioritization costs can differ across techniques). The technique with the lowest C_k will be the most cost-beneficial.

4 Empirical Studies

The foregoing models are meant to help us better evaluate cost benefits tradeoffs involving regression testing techniques. A fundamental research question is whether the new models succeed in this: are they more accurate, comprehensive, or realistic than previous models? In particular, for selection and reduction, we wish to know how the old and new models behave as the ratio of fault cost to test cost changes, and whether the new models assess cost-benefits tradeoffs better than previous models. For prioritization, we wish to know how the new model will assess cost-benefits of prioritization techniques, and how it will rank the techniques.

To gain insight into these issues, we designed and performed three exploratory experiments. In the following sections we describe our variables, experiment materials, and experiment design.

4.1 Variables

For selection and reduction, our experiments manipulated two independent variables: the cost model used to evaluate the methodology and the ratio of fault cost to test cost that the cost models depend on. For prioritization, our experiments manipulated two independent variables: the prioritization technique and the ratio of the cost of a programmer waiting one time unit to analysis cost per test.

The dependent variables for these experiments are the outputs of our models, and include costs of, or savings achieved by, the application of a particular regression testing technique.

Version	Funcs.	Mod'd Funcs.	LOCs	Regr. Faults
2.0	1,494	—	48,292	0
2.01	1,537	296	49,555	9
2.01.1	1,538	44	49,666	7
2.02	1,678	296	58,090	7
2.02.1	1,678	12	58,103	3
2.03	1,703	188	59,010	9
2.04	1,890	339	63,802	5
2.05-beta1	1,942	447	65,477	6
2.05-beta2	1,949	40	65,591	7
2.05	1,950	27	65,632	5

Table 1. Bash Subject.

4.2 Regression testing techniques

As selection techniques we chose *retest-all* and *modified non-core entity*. The retest-all technique is our control technique. The *modified non-core entity* technique [15] selects test cases that exercise functions, in P , that (1) have been deleted or changed in producing P' , or (2) use variables or structures that have been deleted or changed in producing P' , but ignores “core” functions, defined as functions exercised by more than 80% of the test cases in the test suite.

As reduction techniques we chose *no reduction* and *GHS reduction*. The no reduction technique (equivalent to retest-all) acts as our control. The GHS reduction technique is an heuristic presented by Gupta, Harrold, and Soffa [9] that attempts to produce suites that are minimal for a given coverage criterion; we used a function coverage criterion.

As prioritization techniques we chose *random*, *total function coverage*, *additional function coverage*, and *optimal* prioritization. Random prioritization (equivalent to retest-all) places test cases in T in random order and is our control. Total function coverage prioritization [20] orders test cases by decreasing the amount of function coverage. Additional function coverage prioritization [20] iteratively selects a test case that yields the greatest function coverage, then adjusts the coverage information on subsequent test cases to indicate their coverage of functions not yet covered, and then repeats this process, until all functions covered by at least one test case have been covered. Optimal prioritization uses information on which test cases in T reveal faults in P' to find an (approximate) optimal ordering for T .

4.3 Experiment Materials

For these experiments we utilized the `bash` subject program previously set up for experimentation as reported in [15]. `Bash` is a popular shell that provides a command line interface to multiple Unix services [14]. As Table 1 shows, the latest versions of `bash` consist of over 50,000 lines of code and almost 2000 functions.

Given the complexity and cost of preparing such a large subject for experimentation, we have invested significant effort in providing a supporting infrastructure so we can

reuse the `bash` subject to answer different research questions. Part of that infrastructure includes a test suite of 1168 test cases. We created this test suite using two complementary methods. First, we evaluated and refined the test suite that accompanied `bash` release 2.0. (We used the test cases from release 2.0 because they are the only ones that work across all releases.) Second, to exercise functionality not covered by the original test suite, we created additional test cases by considering the reference documentation for `bash` [14] as an informal specification. The resulting test cases exercise an average of 64% of the functions across all the versions of the system.

A second part of the infrastructure consists of a set of seeded faults created by a fault seeding process. Since we wished to evaluate the performance of regression testing techniques with respect to detection of regression faults, we asked several graduate and undergraduate computer science students, each with at least two years experience programming in C and unacquainted with the details of this study, to become familiar with `bash` and to insert regression faults into the program versions. We then determined which faults were exposed by each test case. (Further details about this process can be found in [15].) The numbers of faults utilized in the our experiments are reported in column five of Table 1.

Finally, we developed various tools to perform program instrumentation and various regression testing tasks. We used `Cllic` to instrument and monitor the software and obtain coverage information [7]. We created test case prioritization, test suite reduction, and regression test selection tools implementing the techniques described in Section 4.2. We used Unix utilities and direct inspection to determine modified functions, or functions using modified structures.

4.4 Experiment Design

To assess how our models compared with previous models, we designed three experiments. Each experiment evaluates the research question for selection, reduction, and prioritization, respectively. We next discuss the design of those experiments; the section that follows presents their results.

4.4.1 Experiment 1: Regression test selection

This first experiment compares our model with the model presented in [11], which omits the costs of faults. The models are used to assess the two selection techniques described in Section 4.2.

For each version v of `bash`, we simulate the following regression testing process. While developing version $v + 1$, the source code of version v is analyzed, and data are collected about v 's static and dynamic behavior on test suite T . When version $v + 1$ is ready and regression testing is about to begin, a selection algorithm is applied to select $T' \subseteq T$, T' is executed, and its results are validated.

Applying our cost models to these techniques, for version $v + 1$, the cost of retest-all includes the cost of executing and validating all tests in the original test suite and is $C = (e + c) \times n$ units. The cost of modified non-core entity selection for the same version is $C' = (a + s) \times n + (e + c) \times n' + f \times |F|$ units and $C'' = (a + s) \times n + (e + c) \times n'$ units under the new and old models, where n' is the size of the selected test suite and $|F|$ is the number of faults left undetected due to not selecting all tests. As a simplification, we use uniform test and fault costs.

The second independent variable we manipulate is the ratio of fault to test cost. We vary the ratio $\frac{f}{e+c}$ (letting $e + c$ be equal to one cost unit). As values for a and s , based on values calculated in a previous experiment [2], we assign $a + s = k(e + c)$ where k represents the upper bound constant of proportionality relating $(a + s)$ to $(e + c)$ over multiple runs of selection for a program similar to `bash`.

4.4.2 Experiment 2: Test suite reduction

This experiment compares reduction models for the two reduction algorithms described in Section 4.2. The regression testing process we assume is similar to the one assumed for selection, except the reduced test suite T' is employed subsequently over versions $v + i$ ($i \in [1 \dots g]$).

Applying our cost models to these techniques, for version $v + 1$ and $g - 1$ successive versions, the cost of no reduction includes the cost of executing and validating all tests in the original test suite and is $C_r = g \times (e + c) \times n$ units. The cost of GHS reduction for g iterations of regression testing is $C'_r = g \times (e + c) \times n' + \sum_{i=1}^g (f \times |F_i|)$ units for the new model and $C''_r = g \times (e + c) \times n'$ units for the old model, where n' is the size of the reduced test suite and F_i is the number of faults left undetected on the i -th iteration due to not selecting all tests. We call test cost $(e + c)$ and ignore test suite management cost m .

We performed our experiment on versions 1 through 7 of `bash`, reducing the test suite on each version $v \in [1 \dots 7]$ and using versions v , $v + 1$, and $v + 2$ to evaluate the costs and benefits of the reduction. We vary the ratio $\frac{f}{e+c}$ (letting $e + c$ be equal to one cost unit).

4.4.3 Experiment 3: Prioritization

Our third experiment examines the effectiveness of prioritization cost models for assessing cost benefits tradeoffs of the four prioritization techniques considered. For the new model, to compare savings or costs of techniques, we use the formula $delays^{O'} - delays^{O''} - a|T|$, where O' is random order and O'' is prioritized order.

The previous $APFD_C$ measure allows us to compare techniques, but not to assess relative cost-effectiveness. This experiment, therefore, considers the relative costs of prioritization techniques under the new model, varying the ratio (programmer's cost per time unit to analysis cost per test). We assume that all tests have uniform costs.

4.5 Results and Analysis

Our experiments manipulate cost ratio and regression testing technique to evaluate cost models. Our analysis of results, however, does not focus on showing significant differences between the models, but concentrates on providing graphical evidence to characterize the overall tendencies, while emphasizing the distinctive cases exposing the circumstances that lead to differences in the models.

4.5.1 Experiment 1: Regression test selection

Figure 1 shows the results of applying the two cost models to selection on four fault-cost-to-test-cost ratios. The graphs show the values of $C - C'$ and $C - C''$ across all versions: that is, the difference in the savings achieved by the retest-all and modified non-core entity selection techniques, per version, as predicted by the models. As fault-cost-to-test-cost ratios we chose 1:1, 100:1, 1000:1, and 10000:1; graphs (a) through (d) show results for each. The X axes show versions and the Y axes show the cost savings.

As graphs (a) and (b) show, when ratios are small, the two models generate almost identical results: retest-all and modified non-core entity produce equivalent savings under both models (the graphs show that savings occurred on versions 2, 4, 6, 8, and 9; on others, savings are near 0).

When the ratio is 1000:1, the two models generate noticeably different results, so under the new model, selection becomes less cost effective on versions 2 and 9. This tendency is accentuated when the ratio is 10,000:1, where the models generate drastically different results. Note that, under the old model, selection is shown to be always cost effective; however, under the new model, on versions 2 and 9, selection is shown to be substantially more expensive than retest-all. In these two particular versions, selection avoided the execution of some test cases, but the cost of missing a fault clearly did not make this technique worthwhile.

Implications. If the fault-to-test-cost ratio is sufficiently high, a model omitting the costs of faults can make the non-core selection technique look cost beneficial in cases where it is not. The new model accurately reflects fault cost, reducing the chance of choosing an inappropriate technique.

4.5.2 Experiment 2: Test suite reduction

Figure 2 shows the results of using the two cost models to evaluate the relative effects of reduction and no reduction on `bash`. The graphs show the values of $C_r - C'_r$ and $C_r - C''_r$ across all starting versions (v1 to v7). We use the same ratios as in Experiment 1. The graphs again show that when ratios are small, the two models generate almost identical results: retest-all and reduction produce equivalent savings under both models (savings occurring on all versions).

When the fault-cost-to-test-cost ratio is 1000:1, the new model indicates that reduction is not cost effective for version 1, where the benefits of avoiding the execution of 96% of the tests is surpassed by the costs of 7 undetected faults.

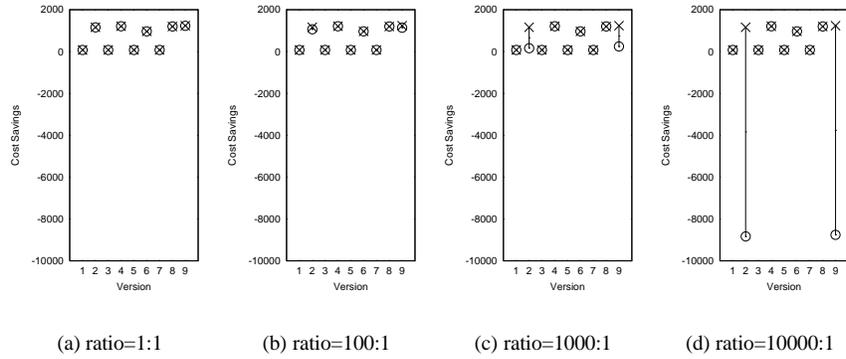


Figure 1. Selection (×: old model, ○: new model, vertical lines: difference between models.)

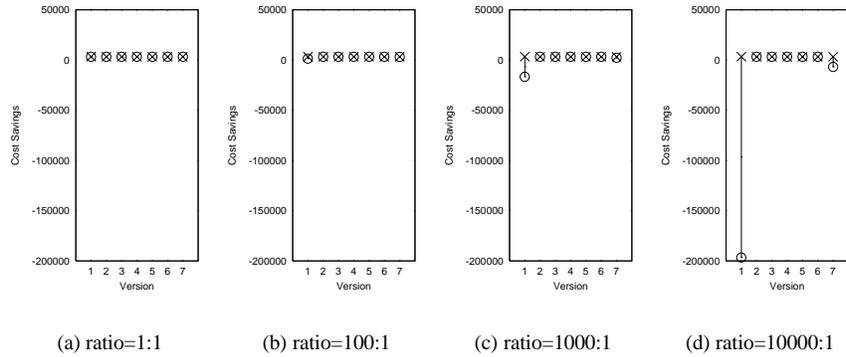


Figure 2. Reduction (×: old model, ○: new model, vertical lines: difference between models.)

When fault-to-test-cost ratio is 10,000:1, the two models generate even more different results, especially for versions 1 and 7.

Implications. As with selection, models not incorporating fault costs will tend to overestimate the cost effectiveness of a reduction technique. This problem is magnified for reduction due to its likelihood of producing a test suite that misses some faults.

4.5.3 Experiment 3: Prioritization

Figures 3(a), 3(b), and 3(c) show the cost-benefits trade-offs among prioritization techniques by varying the relationship between programmer’s cost and analysis cost. The X axes show the `bash` version and the Y axes show the cost $delays^{O'} - delays^{O''} - an$, where O'' is the order of the test suite produced by a given prioritization technique, O' is the order produced by the random method, a is the analysis cost per test, and n is the size of the test suite. Higher values mean higher savings due to prioritization. Different lines correspond to different ratios for ratio of programmer’s costs to selection costs. We can see that, when this ratio is small (1:1, 10:1), there are hardly any savings over random ordering. However, as the ratio increases to 100:1 and 1000:1, the savings become more obvious.

Where prioritization technique is concerned, optimal gives the best results in term of savings. However, optimal

cannot actually be implemented if faults are not known. Additional functional coverage is the next technique to show substantial savings in costs; for example, when the ratio is 10:1, for version 7, the additional function coverage technique saves 2,714,220 cost units. Total function coverage shows more modest cost savings, and on some versions, is not even beneficial relative to random ordering.

Implications. The prioritization techniques considered ranked identically in the presence of varying ratios. However, the new model clearly provides a more accurate assessment of how much practical benefit can be obtained through a prioritization technique – the main consideration when incorporating a technique into a testing process.

5 Conclusions

We have developed cost models for assessing the cost-benefits of regression test selection, test suite reduction, and test case prioritization techniques. These models have been applied to historical data gathered for `bash`.

In this paper, we have primarily considered assessment after the fact of regression testing techniques; this can be used by practitioners to make decisions analyzing historical data, or by researchers to evaluate experimental results. Our next step is to explore the possibility of predicting the cost-effectiveness of techniques in advance; an effort in which the cost models presented in this paper will be instrumental.

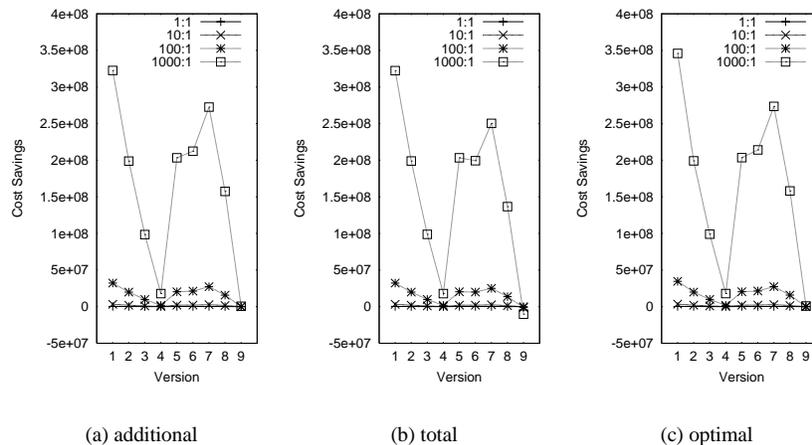


Figure 3. Prioritization: differences between given techniques and random across various cost ratios.

Acknowledgements

This work was supported in part by the NSF ITR program under Awards CCR-0080898 and CCR-0080900 to University of Nebraska, Lincoln and Oregon State University, by NSF Awards CCR-9703108 and CCR-9707792 to Oregon State University, and by an NSF-EPSCOR Grant Award to the University of Nebraska, Lincoln. Praveen Kallakuri helped with the preparation of bash.

References

- [1] B. Beizer. *Software Testing Techniques*. Van Nostrand Reinhold, New York, NY, 1990.
- [2] J. Bible, G. Rothermel, and D. S. Rosenblum. A comparative study of coarse- and fine-grained safe regression test selection techniques. *ACM Trans. Softw. Eng. Meth.*, 10(2):149–183, April 2001.
- [3] T. Y. Chen and M. F. Lau. Dividing strategies for the optimization of a test suite. *Info. Proc. Let.*, 60(3):135–141, Mar. 1996.
- [4] Y.-F. Chen, D. Rosenblum, and K.-P. Vo. Testtube: A system for selective regression testing. In *Proc. 16th Int'l. Conf. Softw. Eng.*, pages 211–220, May 1994.
- [5] S. Elbaum, A. Malishevsky, and G. Rothermel. Incorporating varying test costs and fault severities into test case prioritization. In *Proc. 23rd Int'l. Conf. Softw. Eng.*, pages 329–338, May 2001.
- [6] S. Elbaum, A. Malishevsky, and G. Rothermel. Test case prioritization: A family of empirical studies. *IEEE Trans. Softw. Eng.*, 28(2):159–182, February 2002.
- [7] S. Elbaum, J. Munson, and M. Harrison. CLIC: A tool for the measurement of software system dynamics. In *SETL Technical Report - TR-98-04.*, 04 1998.
- [8] T. L. Graves, M. J. Harrold, J.-M. Kim, A. Porter, and G. Rothermel. An empirical study of regression test selection techniques. *ACM Trans. Softw. Eng. Meth.*, 10(2):184–208, April 2001.
- [9] M. J. Harrold, R. Gupta, and M. L. Soffa. A methodology for controlling the size of a test suite. *ACM Trans. Softw. Eng. Meth.*, 2(3):270–285, July 1993.
- [10] H. K. N. Leung and L. White. Insights into regression testing. In *Conf. Softw. Maint.*, pages 60–69, October 1989.
- [11] H. K. N. Leung and L. White. A cost model to compare regression test strategies. In *Proc. Conf. Softw. Maint.*, pages 201–208, Oct. 1991.
- [12] H. K. N. Leung and L. J. White. A study of integration testing and software regression at the integration level. In *Proc. Conf. Softw. Maint.*, pages 290–300, Nov. 1990.
- [13] J. Offutt, J. Pan, and J. M. Voas. Procedures for reducing the size of coverage-based test sets. In *Proc. Twelfth Int'l. Conf. Testing Computer Softw.*, pages 111–123, June 1995.
- [14] C. Ramey and B. Fox. *Bash Reference Manual*. O'Reilly & Associates, Sebastopol, CA, 2.2 edition, 1998.
- [15] G. Rothermel, S. Elbaum, A. Malishevsky, P. Kallakuri, and B. Davia. The impact of test suite granularity on the cost-effectiveness of regression testing. In *Proc. 24th Int'l. Conf. Softw. Eng.*, pages 130–140, May 2002.
- [16] G. Rothermel and M. Harrold. Analyzing regression test selection techniques. *IEEE Trans. Softw. Eng.*, 22(8):529–551, Aug. 1996.
- [17] G. Rothermel and M. J. Harrold. A safe, efficient regression test selection technique. *ACM Trans. Softw. Eng. Meth.*, 6(2):173–210, April 1997.
- [18] G. Rothermel and M. J. Harrold. Empirical studies of a safe regression test selection technique. *IEEE Transactions on Softw. Eng.*, 24(6):401–419, June 1998.
- [19] G. Rothermel, M. J. Harrold, J. Ostrin, and C. Hong. An empirical study of the effects of minimization on the fault detection capabilities of test suites. In *Proc. Int'l. Conf. Softw. Maint.*, pages 34–43, November 1998.
- [20] G. Rothermel, R. Untch, C. Chu, and M. J. Harrold. Test case prioritization. *IEEE Trans. Softw. Eng.*, 27(10):929–948, October 2001.
- [21] E. W. Wong, J. R. Horgan, S. London, and A. P. Mathur. Effect of test minimization on fault detection effectiveness. In *Proc. Int'l. Conf. Softw. Eng.*, pages 41–50, 1995.
- [22] W. E. Wong, J. R. Horgan, S. London, and H. Agrawal. A study of effective regression testing in practice. In *Proc. Eighth Int'l. Symp. on Softw. Reliability Engineering*, pages 230–238, November 1997.