Predicting the Cost-Effectiveness of Regression Testing Strategies

David S. Rosenblum and Elaine J. Weyuker

AT&T Research
600 Mountain Avenue
Murray Hill, NJ 07974
{dsr,weyuker}@research.att.com

Abstract

Selective regression testing strategies aim at choosing an appropriate subset of test cases from among a previously run test suite for a software system, based on information about the changes made to the system to create new versions. Although there has been a significant amount of research in recent years on the design of such strategies, there has been significantly less investigation of their cost-effectiveness. In this paper some computationally efficient predictors of the cost-effectiveness of the two main classes of selective regression testing approaches are presented. A case study is described in which these predictors are used to assess the appropriateness of using a particular regression testing strategy to test multiple versions of a widely-used software system.

Keywords—cost estimation, regression testing, software analysis, test coverage

1 Introduction

Selective regression testing strategies aim at choosing an appropriate subset of test cases from among a previously run test suite for a software system, based on information about the changes made to the system to create new versions [1, 3, 4, 5, 6, 8, 9, 11, 12, 13]. The motivation for such strategies is the desire to keep the cost of regression testing manageable. The intuition is that if, instead of rerunning the entire test suite (the so-called *retest-all* strategy), a systematically-selected subset is chosen to be rerun, then substantial resources will be saved due to the limited size of the test suite. Although there has been a significant amount of research in recent years on the design of such strategies, there has been significantly less investigation of their cost-effectiveness. The primary exception to this statement is a simple cost model described by Leung and White, to enable the comparison of regression testing strategies [10]. There also has been little investigation of the various factors affecting cost, including CPU time, disk space, effort of testing personnel, the cost of business opportunities gained or lost through increased or reduced testing, and so on.

Rothermel and Harrold group a number of selective regression testing approaches into three categories [12]. *Safe* approaches require the selection of every existing test case that exercises any program element that could be affected by a given program change. *Minimization* approaches attempt to select the smallest set of test cases necessary to test affected program elements at least once. *Coverage* approaches attempt to assure that some structural coverage criterion is met by the test cases that are selected. Because minimization approaches typically attempt to select a minimal subset of test cases *satisfying some coverage criterion*, minimization approaches tend to be special types of coverage approaches. For this reason, for the purposes of this paper, we will only distinguish between approaches that attempt to select *all* affected test cases, and those that attempt to select a *minimal* set of affected test cases.

Many coverage criteria do not actually require that a minimal test set be selected. In a sense, safe strategies and minimization strategies can be thought of as being at the two endpoints of a continuum of strategies. In practice, a tester may be satisfied using near-minimal test sets. The search for small test sets is based on the intuition that repeatedly re-exercising code units during testing is "wasteful". However, the effort needed to minimize the test set can be substantial and therefore may not be worthwhile. Note that in general, most of the selective regression testing strategies that have been described in the literature are independent of any coverage criterion that may have been used to create the original test suite. In fact, regression testers are frequently unaware of how the original test suite was designed.

The main thrust in papers on safe selective regression testing has been to show that by using the proposed algorithm to selectively choose a regression test suite, faults detected by the full test suite are guaranteed to be detected by the selected subset. For minimization methods, it is assumed that it is unlikely for faults to go undetected compared with the straightforward retest-all approach. The fundamental assumption underlying all these selective regression testing strategies is that the cost of the analysis necessary to do the selection is offset by the savings realized by the reduced test set size.

When the cost of running individual test cases is substantial, the size of the test set can be an especially important issue. However, as we shall see, it is not always possible to intelligently select a relatively small subset of the regression test suite, and the cost of the analysis necessary for mak-

SIGSOFT '96 CA, USA
© 1996 ACM 0-89791-797-9/96/0010...$3.50

ing this determination may well offset the savings realized by the reduced test set size. It is this issue that we investigate in this paper. In particular, we discuss the design of *predictors* of cost-effectiveness that we will use to determine whether or not the savings a selective regression testing method achieves through a reduction in test cases is likely to be worth the cost of the analysis needed to achieve this reduction. Our goal is to define relatively inexpensive and simple calculations that can provide such a basis for prediction. In this way, we may be able to prevent the waste of significant analysis costs when the strategy under investigation is likely to select all or most of a test suite for regression testing. In this case, the savings realized by the reduced test set size may well be less than the analysis costs and therefore not worth doing.

2 A Model of Regression Testing

In this section we present a formal model of selective regression testing. Many of the methods that have been described in the literature can be reasonably characterized by this model. We will use this model as a basis for computing our predictors.

Informally, selective regression testing involves the systematic selection of a subset of test cases from a permanent regression test suite. This selection is made each time changes are made to a system under test. These changes might be due to such things as fault removals, planned enhancements, specification changes, porting to new platforms, and so on. The selection is driven by two kinds of analysis: coverage analysis and change analysis. *Coverage analysis* is used to identify the relationship between the test suite and the entities in the system under test that are exercised by the test suite.[1] *Entities* might include such things as statements, branches, functions or definition-use associations. *Change analysis* (also known as *impact analysis*) is used to identify the entities that have been modified or could be affected by any modifications made to the system under test. Test selection involves choosing test cases that cover affected entities. Safe methods select all test cases that cover affected entities, while minimization methods attempt to select the smallest subset of test cases such that each affected entity is covered at least once.

One of the main things that distinguishes the various selective regression testing methods that have been proposed is the choice of the entity or entities on which the coverage and change analysis is performed. This choice affects the level of granularity of the analysis, and hence both the precision and efficiency of the analysis. *Precision* is defined to be the ability of a method to avoid selecting test cases that do not cause the modified program to produce outputs that differ from the original version. *Efficiency* assesses the computational cost and automatability of a selective regression testing strategy. These terms, proposed by Harrold and Rothermel, are two of the five criteria they used to evaluate selective regression testing methods [12]. For instance, TESTTUBE is a system for selective regression testing of C programs in which the entities that are analyzed are function definitions, global variable definitions, type definitions and preprocessor macro definitions [3]. On the other hand, dataflow-oriented approaches (such as the approach described by Ostrand and Weyuker [11]) treat definition-use associations as the entities of interest. Still other methods focus on other kinds of entities, such as statements, code segments, or backward slices from the place of each program edit.

More formally, let P be the system under test, and let S be its specification. Let T be the regression test suite for P, with $|T|$ denoting the size of T. Let M be a selective regression testing method to be used to select a subset of T to test P; M may depend on P, S, T, execution histories for T, and other factors. Let E be the set of entities of the system under test that are analyzed by M. We assume that T and E are non-empty, and that every syntactic element of P belongs to at least one entity in E.

Let $covers_M(t,e)$ be the coverage relation induced by M for P and defined over $T \times E$. We define $covers_M(t,e)$ to be true if and only if the execution of P on test case t causes entity e to be *exercised* at least once. If e is a function or module of P, e is exercised whenever it is invoked. If e is a simple statement, statement condition, definition-use association or other kind of execution subpath of P, e is exercised whenever it is executed. If e is a variable of P, e is exercised whenever it is read or written. If e is a type of P, e is exercised whenever a variable of type e is exercised. If e is a macro of P, e is exercised whenever its expansion is exercised. If e is a slice of P, e is exercised whenever all of its constituent statements are exercised. Appropriate meanings for "exercised" can be defined similarly for other kinds of entities of P.

Let E^C denote the set of covered entities. E^C is defined as follows:

$$E^C = \{e \in E \mid \exists t \in T(covers_M(t,e))\}$$

We use $|E^C|$ to denote the number of covered entities.

It is sometimes convenient to represent $covers_M(t,e)$ by a 0-1 matrix C, whose rows represent elements of T and whose columns represent elements of E^C. We then define element $C_{i,j}$ of C as follows:

$$C_{i,j} = \begin{cases} 1 & \text{if } covers_M(i,j) \\ 0 & \text{otherwise.} \end{cases}$$

We define CC to be the amount of *cumulative coverage* achieved by T (i.e., the total number of ones in the 0-1 matrix):

$$CC = \sum_{i=1}^{|T|} \sum_{j=1}^{|E^C|} C_{i,j}$$

Note that if we were to extend C to include columns for the uncovered entities in E, the cumulative coverage CC would remain unchanged since these additional columns would contain only zeroes.

Let T_M be the subset of T selected by M for P, and let $|T_M|$ denote its size. Formally,

$$T_M = \{t \in T \mid M \text{ selects } t\}$$

Let s_M be the cost per test case of applying M to P to select T_M, and let r be the cost per test case of running P on a test case in T and checking its result. Leung and White show that the following relationship must hold in order for M to be a cost-effective method of selecting test cases [10]:[2]

$$s_M\,|T_M| < r\,(|T| - |T_M|)$$

[1] Note that this coverage analysis may be different from any coverage analysis that was used to create the test suite initially.

[2] Note that we have changed the notation and representation used by Leung and White.

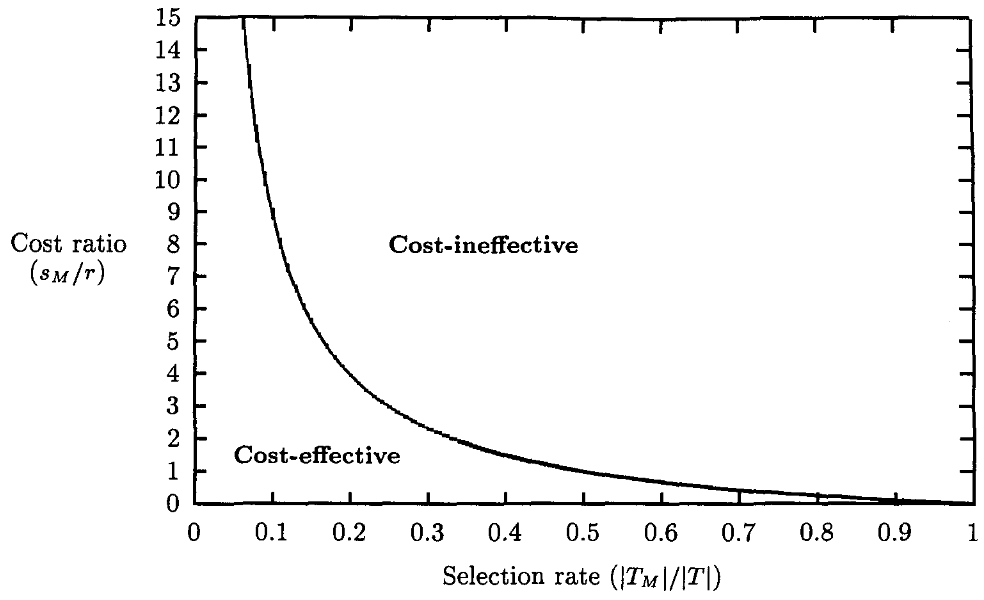

Figure 1: The Leung-White Cost-Effectiveness Curve.

That is, the cost of the analysis needed to select T_M should be less than the cost of running the unselected test cases, which form the set $T - T_M$. This relationship is captured in the cost-effectiveness curve of Figure 1, in which the *selection rate* $|T_M|/|T|$ is plotted along the x-axis, and the *cost ratio* s_M/r is plotted along the y-axis. The curve appearing in the plot is the break-even curve—points lying below the curve represent cost-effective situations, while points lying above the curve represent cost-ineffective situations. For example, if the selection rate is 50%, for every test case that is selected, one test case is eliminated, and thus the cost of selecting one test case must be less than the cost of running one eliminated test case. If the selection rate is 25%, three test cases are eliminated for every one that is selected, and thus the cost of selecting the one test case must be less than the cost of running the three eliminated test cases (i.e., less than three times the cost of running one eliminated test case).

Although we will use the Leung-White cost model to illustrate the application of our predictors, it is important to notice that the Leung-White model makes a number of simplifying assumptions that may well be inappropriate for large systems with certain characteristics. For instance, their model assumes that the costs s_M and r are *constant* for all test cases. Even though such an assumption does make it feasible to apply the model, it is surely not accurate, and empirical research is needed to consider the effects of this simplification. More importantly, it is assumed that all costs can be quantified in terms of simple dollar (or other numeric) figures and are therefore interchangeable and uniform throughout. Again this is not in general a realistic assumption, and more empirical research is needed to assess the effects of this assumption as well.

Rothermel and Harrold suggest that it is useful to divide regression testing into two phases for the purpose of cost analysis. During the *preliminary phase*, changes are made to the software, and the new version of the software is built. During the *critical phase*, the new version of the software is tested prior to its release to customers [12]. When using a selective regression testing method, it is important to do as much analysis as possible during the preliminary phase, allowing the critical phase to be devoted to running test cases as much as possible in order to reduce the likelihood of a delayed release. For instance, for some large projects in development organizations that we work with, the use of specialized, very expensive equipment requires that most testing be performed using a specially-equipped test laboratory that is shared by many different organizations and scheduled well in advance of its expected use. Such a laboratory typically runs continuously, and may be available to a given project no more than a few hours per week, typically during the middle of the night. As such, it is viewed as a scarce resource that is never to be wasted. However, certain types of analysis could be performed during off-hours using spare cycles. In such cases, any analysis that can be performed on individual workstations or PCs in advance of testing may well be more cost-effective and feasible than use of the test laboratory, even if the cost-effectiveness relationship of the Leung-White model is not strictly satisfied. Thus, there may be cases in which even though the apparent cost of analysis seems not to be worth the small number of test cases eliminated by a selective regression testing method, the analysis is nonetheless worthwhile since different resources are used to make the determination, and any savings in test lab use is especially important.

To properly evaluate the cost-effectiveness of a selective regression testing method for such a system, the cost model would therefore need to balance the high cost of running test cases in the test lab against the relatively cheap cost of doing analysis on idle processors. In general, it may be necessary to employ system-specific cost models, taking into account all of the unique characteristics of the system under test, the test suite, and the process and resources used to test the system.

3 Coverage-Based Predictors

In this section we present simple predictors of the number of test cases that need to be rerun when a change is made to the system under test, and we examine their strengths and weaknesses. We also consider situations that affect their applicability.

The design of our predictors is based on some fundamental assumptions about the nature of test coverage and also the nature and distribution of changes made to a system. It is our experience that the relation $covers_M(t, e)$ changes very little during maintenance, except when new, large subsystems or features are added to P, thereby causing large numbers of test cases to be added to T. We also observe that the ability of a method M to eliminate test cases from a test suite T for a system P will be fundamentally governed by the nature of the relation $covers_M(t, e)$. For instance, if there is a great deal of overlap in the sets of entities of P that each test case covers, then we would not expect a safe strategy to be able to eliminate very many test cases, regardless of the degree to which the system is changed from version to version. This is because each entity is covered by a relatively large fraction of the test cases in the test suite, and each of these test cases would need to be rerun. Thus, according to the Leung-White model, the presence of a great deal of overlap of this kind would require an analysis cost that is far less than the cost of running test cases in order for the analysis to be worthwhile.

3.1 Safe Strategies, Single Entity Changed

In attempting to predict whether or not a selective regression testing method M will be cost-effective when employed during the maintenance of software system P, we will first try to predict whether M will be cost-effective when only a single entity of P is changed. If M is not even cost-effective for this smallest possible change to P, then we are guaranteed that M will never be cost-effective for more complex changes to P. If, however, M is found to be cost-effective for testing a change to a single entity of P, then other, more accurate predictors must be employed to determine the limits and circumstances of M's cost-effectiveness.

As a first step in computing the desired predictor, we consider the expected number of test cases that would have to be rerun provided only a single entity has been changed and that a safe method M is being used. This number is simply the average number of test cases that exercise an entity, which can be computed by taking the ratio of the cumulative coverage CC to the size of the entity set E. Call this average N_M:

$$N_M = \frac{CC}{|E|}$$

The intuition here is that if we can expect to have to rerun almost the entire test suite, there may be little or no advantage to doing whatever analysis is necessary to select that subset of the test suite unless the cost of running a test is very high relative to the cost of the analysis. We emphasize that this predictor is only intended to be used when the regression testing strategy's goal is to rerun *all affected* test cases. This approximation is therefore intended to be used only for safe strategies.

A slightly refined variant of N_M considers E^C rather than E as the universe of entities. That is, to determine whether M is cost-effective for testing a single change to P, we compute the average number of test cases that cover each *covered* entity, namely N_M^C:

$$N_M^C = \frac{CC}{|E^C|}$$

Then the fraction of the test suite that needs to be rerun is π_M, which is our predictor for $|T_M|/|T|$:

$$\begin{aligned}\pi_M &= \frac{N_M^C}{|T|} \\ &= \frac{CC}{|E^C||T|}\end{aligned}$$

Thus, if a change is made to one "typical" entity, then N_M^C test cases (or π_M of the test suite) will be needed to test the change. Restricting the computation of N_M^C to only the covered entities produces a more conservative estimate than the estimate that results from considering *all* entities, as was done for N_M above. In particular, the use of N_M^C is predicated on the assumption that a change is made to a covered entity and that therefore at least one test case will be selected.

In order to compute π_M, it is first necessary to carry out the same coverage analysis that would be used when applying M during maintenance. When performing the coverage analysis, the cost of the analysis can be computed, as can the cost r of running a test case. This gives a lower bound on s_M, the per-test-case cost of applying M (the other component of the cost being the cost of the change analysis). Call this estimate of the per-test-case coverage analysis cost $s_{M,C}$. Once π_M and $s_{M,C}$ have been computed, they can then be plugged into the Leung-White cost model, along with the estimate of r, to determine whether or not M lies within the cost-effectiveness range for selection rate π_M. If it does not, then M can be ruled out for the given software system; if it does, then more accurate estimates must be computed for multiple entity changes.

It may seem at first that applying method M (or a portion of M) to determine whether M is cost-effective defeats the whole purpose of avoiding the cost of using M when it is not cost-effective. However, as mentioned above, we expect the coverage relationship between T and P to change very little from version to version of P (at least to the extent that this relationship affects the value of π_M). Therefore, by carrying out part of M on just one version of P to compute π_M, it may be possible to determine that it will not be cost-effective to employ M on *any* version of P. In that case, the costs of using M can be avoided in all future versions of P, even though the cost is incurred for only one version of P.

Note that using averages in the computation of N_M^C ignores the fact that there can be wide variation in the amount of change various entities incur. Indeed, because these predictors are averages, their accuracy might vary significantly. For example, consider the three coverage patterns shown in Figures 2, 3 and 4. For the pattern A, shown in Figure 2, with a safe selection strategy, two test cases will always need to be rerun after a change to a single entity, regardless of which entity is changed. Since the test set contains $|E^C|$ test cases, $2/|E^C|$ represents the fraction of the test suite that will be selected for any changed entity, and this is exactly what π_M predicts.

Next consider the pattern B, shown in Figure 3. In that case, with a safe strategy, exactly two test cases will also

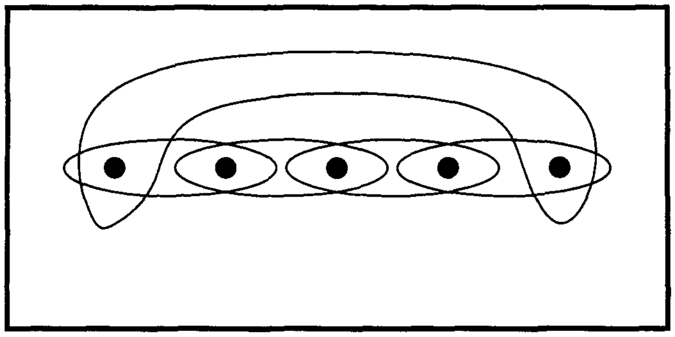

Figure 2: Coverage Pattern A.

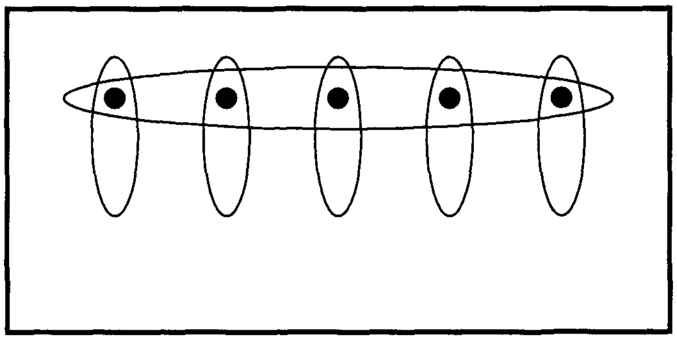

Figure 3: Coverage Pattern B.

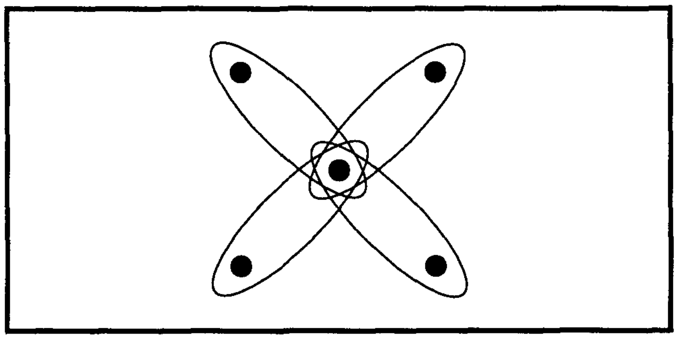

Figure 4: Coverage Pattern C.

be rerun regardless of which entity is changed, but now the test set size is $|E^C|+1$. So we expect, on average, to have to rerun $2/(|E^C|+1)$ of the test suite. Again this is exactly what π_M predicts.

Finally, consider pattern C, shown in Figure 4, in which each test case covers the "core" entity shown in the center, plus one of the other "non-core" entities. In this pattern, there are $|E^C|-1$ test cases. π_M predicts the value $2/|E^C|$ as the fraction of the test suite needed to test a change to a single entity. If any of the non-core entities is changed, then only one test case needs to be rerun, and the fraction required would be slightly less than the value predicted (assuming a large value of $|E^C|$). On the other hand, if the core entity is changed, then all test cases would be selected. Hence, in no case does π_M correctly predict the *exact* number of test cases that will be selected, and in one case, the prediction significantly underestimates the number of test cases that must be rerun. Still, if each entity is equally likely to be changed, then over *all* future versions in maintenance, π_M accurately predicts the *average* fraction of the test suite selected in any one version.

These scenarios demonstrate the limitations of employing simple averages to compute the desired predictors. Nonetheless, π_M does provide us with an estimate that at least can be used to rule out M as a cost-effective strategy. In addition, if information is available about the frequency with which changes are made to the different entities in the system under test, then this information can be used to weight the averages computed in π_M.

3.2 Safe Strategies, Multiple Entities Changed

If the computation of π_M predicts that M *will* be cost-effective in the presence of a change to a single entity, then changes to multiple entities must be considered, since typical software changes will involve several entities.

As a point of departure, we can generalize the computation of N_M^C and π_M to predict the number of test cases needed to test changes to multiple entities in coverage patterns A, B and C. For this discussion, let k be the number of entities changed, where

$$1 \leq k \leq |E^C|$$

Thus, there are $\binom{|E^C|}{k}$ ways of selecting k entities from a set containing $|E^C|$ entities, assuming that each entity is equally likely to be changed. We will typically assume that there are a relatively large set of entities and a relatively small number of changes k. We therefore assume that

$$k \ll |E^C|$$

For pattern A shown in Figure 2, the number of test cases needed will range between $k+1$ (if all of the changed entities are "adjacent" to each other, and therefore "share" test cases) and $2k$ (if none of the changed entities are "adjacent" to each other, and are therefore unable to "share" test cases).

For pattern B shown in Figure 3, the number of test cases needed will always be $k+1$ because every entity is exercised by a test case that exercises only that entity, as well as by the test case that exercises all entities. Thus, N_M^C increases linearly with k for this pattern.

For pattern C shown in Figure 4, the number of test cases needed will be either k if the core entity is *not* changed, or $|E^C|-1$ if the core entity *is* changed. We can compute N_M^C as the sum of these two values weighted by their frequency of occurrence:

$$\begin{aligned} N_M^C &= \frac{k\binom{|E^C|-1}{k}}{\binom{|E^C|}{k}} + \frac{(|E^C|-1)\binom{|E^C|-1}{k-1}}{\binom{|E^C|}{k}} \\ &= \frac{k(|E^C|-k)}{|E^C|} + \frac{(|E^C|-1)k}{|E^C|} \\ &= \frac{k}{|E^C|}\left(2|E^C| - k - 1\right) \end{aligned}$$

We refer to the factor $k/|E^C|$ as the *change rate*, the fraction of covered entities that are changed.

We can compute π_M as before by dividing N_M^C by $|T|$, which for pattern C is equal to $|E^C|-1$:

$$\begin{aligned} \pi_M &= \frac{N_M^C}{|T|} \\ &= \frac{k}{|E^C|}\frac{\left(2|E^C| - k - 1\right)}{|E^C|-1} \end{aligned}$$

We note that when k is 1, π_M is $2/|E^C|$, which is the value computed using the formula for π_M presented in Section 3.1.

For large values of $|E^C|$, we can approximate π_M as follows:

$$\begin{aligned} \pi_M &\approx \frac{k}{|E^C|}\frac{\left(2|E^C| - k - 1\right)}{|E^C|} \\ &\approx 2\left(\frac{k}{|E^C|}\right) - \left(\frac{k}{|E^C|}\right)^2 \end{aligned}$$

Figure 5 presents a plot of the selection rate π_M versus the change rate $k/|E^C|$ for pattern C. As the plot shows, π_M increases rapidly as k increases, and hence as the the change rate increases.

As was shown in Section 3.1, we can also compute estimates for r and $s_{M,C}$ and use the Leung-White model to analyze the cost-effectiveness of M. In particular, the estimates for r and $s_{M,C}$ can be plugged into the Leung-White model to determine the maximum cost-effective value for the selection rate π_M. This maximum can then be used to find the maximum number of changed entities k for which cost-effectiveness can still be achieved. The feasibility of achieving such a change rate can then be determined by consulting with testing personnel or through an analysis of historical change data for P.

3.3 Minimization Strategies

Minimization strategies have the special property that a change to a single entity requires that only one test case be rerun regardless of the nature of the relation $covers_M(t, e)$. Therefore, we focus on the more interesting situation in which multiple entities are changed. We consider next the case in which k entities are changed.

For pattern A shown in Figure 2, the number of test cases needed when a minimization selection strategy is used will

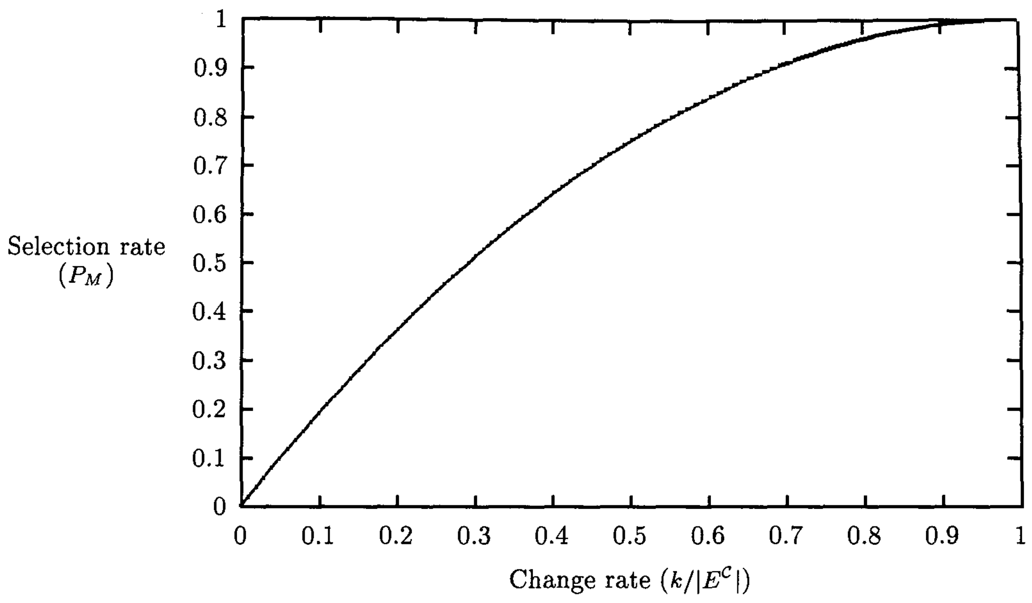

Figure 5: Plot of π_M versus $k/|E^C|$ for Coverage Pattern C of Figure 4 (Safe Strategies).

range between $k/2$ and k, again depending on the "proximity" of the changed entities.

For pattern B shown in Figure 3, the number of test cases needed will always be 1, since the test case that exercises all of the entities can be selected.

For pattern C shown in Figure 4, the number of test cases needed will be either $k-1$ if the core entity is *not* changed, or k if the core entity *is* changed. As in Section 3.2, we can compute N_M^C as the sum of these two values weighted by their frequency of occurrence:

$$\begin{aligned} N_M^C &= \frac{k\binom{|E^C|-1}{k}}{\binom{|E^C|}{k}} + \frac{(k-1)\binom{|E^C|-1}{k-1}}{\binom{|E^C|}{k}} \\ &= \frac{k(|E^C|-k)}{|E^C|} + \frac{(k-1)k}{|E^C|} \\ &= \frac{k}{|E^C|}(|E^C|-1) \end{aligned}$$

As before, normalizing N_M^C by dividing by $|T|$, we get

$$\begin{aligned} \pi_M &= \frac{N_M^C}{|T|} \\ &= \frac{k}{|E^C|}\frac{|E^C|-1}{|E^C|-1} \\ &= \frac{k}{|E^C|} \end{aligned}$$

This last value is simply the change rate, i.e., the fraction of the entity set that is changed. Thus, when minimization strategies are applied to pattern C, the selection rate is equal to the change rate, and the number of test cases needed is simply k, the number of entities changed.

Estimates for π_M can then be plugged into the Leung-White model as was discussed in Section 3.2, along with estimates for r and $s_{M,C}$.

4 Experience

When new test selection strategies of any kind are proposed, it is generally necessary to assess their effectiveness and cost. A safe, code-based selective regression testing approach described by Chen, Rosenblum and Vo, called TESTTUBE, was designed explicitly to address the issue of cost-effectiveness [3]. In order to assess the usefulness of the approach, a case study was performed on 31 versions of the 1988 release of the KornShell, a command processor for the UNIX® operating system [2].[3] It was found that in only six versions (20%, with the first version ignored since all test cases were necessarily run on the first version prior to analysis), at least one test case was eliminated from the entire test set using the TESTTUBE approach; in three of these six versions no test cases were selected, either because the versions involved changes to documentation or to uncovered code. This means that in the remaining 24 versions (80%) the analysis required by the approach was entirely wasted. Even in those versions for which less than the entire test suite was selected for retesting, the savings that was achieved was less than the cost of the analysis. This lack of cost-effectiveness may be attributed to a number of factors—the coarse level of granularity of TESTTUBE's analysis, the very low cost of executing the KornShell test suite, and the structure and behavior of the KornShell itself. This last factor arises from the fact that the KornShell is essentially a language processing system, all of whose subsystems (input/output components, tokenizer, parser, semantic analyzer and command execution component) are invoked on even the smallest of inputs. Therefore, changes made to the KornShell frequently impact all subsystems of the software.

In this case study, an average of 88.1% percent of the test suite was selected by TESTTUBE to retest each version. If we had instead used our predictor to determine whether or not we could expect TESTTUBE to be cost-effective for

[3]UNIX is a registered trademark licensed exclusively by Novell, Inc.

selecting test cases for KornShell, we would have computed a value of 87.3% for π_M for a change to one entity in the first version, which is very close to the actual average of 88.1%. This indicates that there is a great deal of overlap in the entities covered by the KornShell test suites, leading us to conclude that there is very limited opportunity for the TESTTUBE method to eliminate test cases during maintenance of the KornShell. Furthermore, computing π_M for all 31 versions reveals that the percentage varies very little over the version history. For the 31 versions we considered, π_M ranged between 82.5% and 87.5%, providing some confirmation for the hypothesis that the relation $covers_M(t, e)$ is relatively stable throughout maintenance.

The average cost r of executing a KornShell test case was 0.052 CPU-minutes on a Sun SPARCstation 1+ workstation, and the per-test-case cost $s_{M,C}$ of carrying out TESTTUBE's coverage analysis was 0.765 CPU-minutes.[4] Applying the Leung-White cost-effectiveness model to the predicted selection rate and test execution cost indicates that the total per-test-case analysis cost s_M must be less than 0.014 CPU-minutes. Clearly, using this measure, TESTTUBE would not be considered cost-effective for KornShell.

As a further test of the predictors, the cost-effectiveness of two selective regression testing methods were analyzed for possible use on a single version of SFIO, an I/O programming library for the UNIX operating system [7]. One method was TESTTUBE, while the other was a hypothetical safe method employing coverage and change analysis at the statement level. To simulate the coverage analysis employed by the second method, the commercial test coverage tool PureCoverage was used.[5] The per-test-case execution cost r of an SFIO test case was 0.126 CPU-minutes or 0.162 clock-minutes on a Sun SPARCstation 1+ workstation.[6]

For TESTTUBE, π_M was estimated to be 50.9% for a change to a single entity, and the per-test-case coverage analysis cost $s_{M,C}$ was 0.228 CPU-minutes (0.244 clock-minutes). The Leung-White model requires the total analysis cost s_M to be less than 0.122 CPU-minutes (0.156 clock-minutes), again indicating that TESTTUBE would not be cost-effective.

For the hypothetical statement-based method, π_M was estimated to be 21.6% for a change to a single entity, while the per-test-case coverage analysis cost $s_{M,C}$ was 0.179 CPU-minutes (0.487 clock-minutes). Because of the lower selection rate, the Leung-White model places a higher limit of 0.457 CPU-minutes (0.588 clock-minutes) on the total analysis cost s_M, suggesting that this method *may* be cost-effective if statement-level change analysis could be performed efficiently enough. Further analysis of the statement-level coverage relation would be needed to determine how many statements could be changed before the method becomes cost-ineffective.

5 Conclusions

We have described the use of computationally efficient predictors as a way of determining whether or not a tester should consider applying a selective regression testing strategy, given that it may involve considerable resources to perform the analysis necessary to use the strategy. As an initial step in this direction, we have proposed a simple, computationally efficient predictor and applied it to the 1988 release of the KornShell. Our predictor computed that 87.3% of the the test suite would need to be rerun if a single entity were to be changed in this first version. Coupling this prediction with the Leung-White cost model, and using empirically determined values for the average cost of executing a KornShell test case and TESTTUBE's cost of performing coverage analysis, we could have decided *a priori* that it would not have been cost effective to use the TESTTUBE approach to do selective regression testing for KornShell. We also applied our predictors to evaluate the cost-effectiveness of two selective regression testing strategies for a programming library called SFIO.

In order to illustrate the computation of our predictors for changes to multiple entities, we used three sample coverage patterns. While these patterns are not necessarily representative of actual coverage patterns that occur in real software systems, they do help to demonstrate the range of behavior in cost-effectiveness that might be encountered in practice. Further empirical research is needed to identify a comprehensive set of representative coverage patterns so that our results can be generalized to a wide variety of software systems. In addition, we expect to continue developing more sophisticated predictors that can be used to determine the applicability of a proposed selective regression testing strategy for testing a given software system.

Based on our results, we can state that, in general, the cost-effectiveness of a selective regression testing method can never be taken for granted. Much more empirical study is needed to evaluate the various approaches that have been proposed in order to determine the circumstances under which they will be cost-effective.

[4]SPARC is a trademark of SPARC International, Inc.

[5]PureCoverage is a trademark of Pure Software.

[6]For the SFIO case study, we computed both CPU times and clock times. In general, clock time may be a better measure of execution cost than CPU time, since test execution often involves a great deal of manual activity that is not performed on the computer.

References

[1] P. Benedusi, A. Cimitile, and U. DeCarlini. Post-maintenance Testing Based on Path Change Analysis. In *Proc. Conf. Software Maintenance 1988*, Phoenix, Oct. 1988, pp. 352–361.

[2] M.I. Bolsky and D.G. Korn, *The New KornShell Command and Programming Language*, Prentice-Hall, 1995.

[3] Y.-F. Chen, D.S. Rosenblum and K.-P. Vo. TestTube: A System for Selective Regression Testing. In *Proc. 16th Int'l. Conf. on Software Engineering* Sorrento, Italy, May 1994, pp. 211–220.

[4] K.F. Fischer. A Test Case Selection Method for the Validation of Software Maintenance Modifications. In *Proc. COMPSAC '77*, Nov. 1977, pp. 421–426.

[5] M.J. Harrold and M.L. Soffa. An Incremental Approach to Unit Testing during Maintenance. In *Proc. Conf. Software Maintenance 1988*, Phoenix, Oct. 1988, pp. 362–367.

[6] J. Hartmann and D.J. Robson. Techniques for Selective Revalidation. *IEEE Software*, Jan. 1990, pp. 31–36.

[7] D. G. Korn and K.-P. Vo. SFIO: Safe/fast string/file IO. In *Proc. Summer 1991 Usenix Conf.*, pages 235–256. USENIX Association, June 1991.

[8] H.K.N. Leung and L. White. Insights into Regression Testing. In *Proc. Conf. Software Maintenance 1989*, Miami, Oct. 1989, pp. 60–69.

[9] H.K.N. Leung and L. White. Insights into Testing and Regression Testing Global Variables. *Software Maintenance: Research and Practice*, Vol. 2, 1990, pp. 209–222.

[10] H.K.N. Leung and L. White. A Cost Model to Compare Regression Test Strategies. In *Proc. Conf. Software Maintenance 1991*, Sorrento, Italy, Oct. 1991, pp. 201–208.

[11] T.J. Ostrand and E.J. Weyuker. Using Data Flow Analysis for Regression Testing. In *Proc. Sixth Annual Pacific Northwest Software Quality Conf.*, Portland, Or, Sep. 1988, pp. 233–247.

[12] G. Rothermel and M.J. Harrold. A Framework for Evaluating Regression Test Selection Techniques. *Proc. 16th Int'l. Conf. on Software Engineering*, Sorrento, Italy, May 1994, pp. 201–210.

[13] A. Taha, S.M. Thebaut, and S-S. Liu. An Approach to Software Fault Localization and Revalidation Based on Incremental Data Flow Analysis. In *Proc. COMPSAC89*, 1989, pp. 527–534.