Using Mutation Analysis for Assessing and Comparing Testing Coverage Criteria

James H. Andrews, *Member*, *IEEE*, Lionel C. Briand, *Senior Member*, *IEEE*,
Yvan Labiche, *Member*, *IEEE*, and Akbar Siami Namin

Abstract—The empirical assessment of test techniques plays an important role in software testing research. One common practice is to seed faults in subject software, either manually or by using a program that generates all possible mutants based on a set of mutation operators. The latter allows the systematic, repeatable seeding of large numbers of faults, thus facilitating the statistical analysis of fault detection effectiveness of test suites; however, we do not know whether empirical results obtained this way lead to valid, representative conclusions. Focusing on four common control and data flow criteria (Block, Decision, C-Use, and P-Use), this paper investigates this important issue based on a middle size industrial program with a comprehensive pool of test cases and known faults. Based on the data available thus far, the results are very consistent across the investigated criteria as they show that the use of mutation operators is yielding trustworthy results: Generated mutants can be used to predict the detection effectiveness of real faults. Applying such a mutation analysis, we then investigate the relative cost and effectiveness of the above-mentioned criteria by revisiting fundamental questions regarding the relationships between fault detection, test suite size, and control/data flow coverage. Although such questions have been partially investigated in previous studies, we can use a large number of mutants, which helps decrease the impact of random variation in our analysis and allows us to use a different analysis approach. Our results are then compared with published studies, plausible reasons for the differences are provided, and the research leads us to suggest a way to tune the mutation analysis process to possible differences in fault detection probabilities in a specific environment.

Index Terms—Testing and debugging, testing strategies, test coverage of code, experimental design.

1 Introduction

Experimentation is an essential part of research in software testing. Typically, experiments are used to determine which of two or more methods is superior for performing some testing-related activity. For instance, one may be interested in comparing the fault detection effectiveness of several testing criteria used to derive test cases and one resorts to experiments to that aim. Testing experiments often require a set of subject programs with known faults. These subject programs should be big enough to be realistic, but not so big as to make experimentation infeasible. As for the faults, the ability of a technique to deal with the given faults should be an accurate predictor of the performance of the technique on real faults.

One problem in the design of testing experiments is that real programs of appropriate size with real faults are hard to find and hard to prepare appropriately (for instance, by preparing correct and faulty versions). Even when actual programs with actual faults are available, often these faults are not numerous enough to allow the experimental results to achieve statistical significance. Many researchers therefore have taken the approach of introducing faults into correct programs to produce faulty versions.

These faults can be introduced by hand (the experimenter can, for instance, ask experienced engineers to do that) or by automatically generating variants of the code. Generally, we view an automatically generated variant as the result of applying an operator to the code. The operators used in this way are called *mutation operators*, the resulting faulty versions are called *mutants*, and the general technique is called *mutation* or *mutant generation*. The process of analyzing when mutants fail and which test suites trigger such failures is referred to as mutation analysis.

The main potential advantage of mutant generation is that the mutation operators can be described precisely and thus provide a well-defined fault-seeding process. This transparency helps researchers replicate others' experiments, a necessary condition for good experimental science. While hand-introduced faults can be argued to be more realistic, ultimately it is a subjective judgment whether a given fault is realistic or not. Another important advantage of mutant generation is that a potentially large number of mutants can be generated, increasing the statistical significance of results obtained.

However, an important question remains. How do we know whether the ability to detect (or "kill") mutants is an accurate predictor of actual performance, that is, what is the external validity of mutants in assessing the fault detection effectiveness of testing techniques? An answer to this question would have important consequences on how we perform testing experiments and, therefore, on the validity of experimental results.

- *J.H. Andrews and A.S. Namin are with the Computer Science Department, University of Western Ontario, London, Ontario, Canada N6A 5B7. E-mail: {andrews, asiamina}@csd.uwo.ca.*
- *L.C. Briand is with the Simula Research Laboratory, Department of Software Engineering, Martin Linges v 17, Fornebu, PO Box 134, 1325 Lysaker, Norway. E-mail: briand@simula.no.*
- *Y. Labiche is with the Software Quality Engineering Laboratory, Carleton University, 1125 Colonel By Drive, Ottawa, Ontario, Canada K1S 5B6. E-mail: labiche@sce.carleton.ca.*

Manuscript received 2 Mar. 2006; revised 5 June 2006; accepted 11 July 2006; published online 7 Sept. 2006.
Recommended for acceptance by Griswold and B. Nuseibeh.
For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number TSE-0051-0306.

0098-5589/06/$20.00 © 2006 IEEE Published by the IEEE Computer Society

In [1], we studied this question by comparing the fault detection ability of randomly selected test suites on hand-seeded, automatically generated, and real-world faults. Our subject programs were a widely used set of programs with hand-seeded faults and a widely used program with real faults. We generated mutants from the subject programs using a set of standard mutation operators from the literature on mutation testing. Our results suggested that the generated mutants are similar to the real faults but different from the hand-seeded faults and that the hand-seeded faults are harder to detect than the real faults. However, since the test suites we used were randomly selected, questions remained as to whether the results would extend to more realistic testing scenarios, such as when a test engineer selects test suites in order to achieve given coverage criteria.

In this paper, we extend our previous work by similarly comparing the behavior of test suites on mutants and real faults, but this time with test suites selected in order to achieve given levels of one of four standard test coverage criteria: Block, Decision, C-Use, and P-Use coverage [27]. We show that our previous results do extend to these testing scenarios, i.e., that the behavior of these new test suites on the mutants closely resembles their behavior on the real faults. Based on this result, we then investigate the relative cost and effectiveness of the criteria by revisiting fundamental questions regarding the relationships between fault detection, test suite size, and control/data flow coverage [11], [13], [17]. Given the practical importance of structural coverage criteria [3], it is surprising that we still know very little regarding their cost-effectiveness. Although such questions have been partially investigated in previous studies [11], [13], [17], we can use a large number of mutant programs, which helps decrease the impact of random variation in our analysis and allows us to use a different analysis approach. Our analysis procedure is precisely presented and justified so as to allow replication in future studies. We compare our results with published studies and provide plausible reasons for differences between them. This process leads us to suggesting a way to tune the mutation analysis process to a specific environment's fault detection pattern, e.g., detection probabilities. For instance, in an environment where coverage-based test suite construction is applied only after extensive code reviews have been conducted, we would expect only harder faults to remain in the code. We can tune the mutation analysis to this pattern by selecting only an appropriate subset of the hardest mutants.

The paper is structured as follows: Section 2 presents the related work that is directly relevant to our research questions. Section 3 describes all the important aspects of our experiment, including a precise definition of research questions. Section 4 presents the analysis results for each research question. Section 5 concludes the paper by summarizing the main results and presenting future work directions.

2 Related Work

This section is separated into two parts. The first one focuses on the use of mutation analysis in testing research. The second one summarizes the main experimental results regarding the cost and effectiveness of control and data flow coverage criteria.

2.1 Mutation Analysis

The idea of using mutants to measure test suite adequacy was originally proposed by DeMillo et al. [8] and Hamlet [15], and explored extensively by Offutt and Untch [25]. Offutt [22] showed empirical support for one of the basic premises of mutation testing, that a test data set that detects simple faults (such as those introduced by mutation) will detect complex faults, i.e., the combination of several simple faults.

Experiments using faulty variants of programs have been carried out by Frankl and Weiss [12], Thévenod-Fosse et al. [30], and Hutchins et al. [17], and, since then, by many researchers. Frankl and Weiss used nine Pascal programs with one existing fault each, whereas Hutchins et al. hand-seeded 130 faults over the seven programs used. Thévenod-Fosse et al. automatically seeded faults in four small C programs using mutation operators. Generally, these experiments follow the pattern of generating a large "test pool" of test cases, running all the faulty versions on all the test cases in the test pool, observing which test cases detected which faults, and using that data to deduce the fault detection abilities of given test suites drawn from the pool (e.g., test suites that satisfy specific coverage criteria).

Although mutant generation was originally proposed as part of a testing strategy, note that Thévenod-Fosse et al. used it instead as a method for generating faulty versions for experiments. Other researchers who have done this include Kim et al. [18], Memon et al. [21], Andrews and Zhang [2], and Briand et al. [4].

Finally, Chen et al. [7] used both hand-seeded faults and generated mutants in their experiments. They point out that the hand-seeded faults are only a subset of the possible faults and raise the issue of whether the faults were representative, but, as this was not the focus of their research, they do not explore it further.

To conclude, except for our previous study [1], there has been no empirical study that has directly assessed the use of mutants by comparing them with results obtained on real faults. Mutation analysis is, however, commonly used throughout experimental testing research.

2.2 Experimental Results on Control and Data Flow Coverage Criteria

Frankl and Weiss [12] performed the first empirical study where the All-uses and Decision (All-edges) criteria are compared to each other and to the null criterion (random test suites). This is performed on nine very small programs whose size ranges from 33 to 60 LOCs for which the authors had access to real faults (one fault per program was used). Comparisons are done using hypothesis testing using the proportion of adequate test suites that expose a specific fault as a dependent variable. In order to determine whether the differences in the effectiveness of criteria were mostly due to differences in the sizes of adequate test suites, they performed their analysis in two different ways: 1) performing the analysis on all adequate test suites and 2) grouping test suites according to their size (intervals) and performing the analysis on each group. Test suites were generated based on a large test pool developed for each subject program. Then, logistic regression was used to model the relationship between the probability of finding a fault and two covariates: coverage level and test suite size. Results showed that All-uses was not always more effective than Decision and the null criterion, but that, when it was more effective, it usually was much more so. For Decision, when more effective than the null criterion, the difference was much smaller. In other words, results varied according to the program and fault analyzed. Logistic regression also showed varied results as there was only a "clear dependence" of fault detection effectiveness on coverage level for three and four of the programs for All-uses and Decision, respectively.

Hutchins et al. [17] reported that fault detection for both DU (Def-Use), a variant of All Uses [27], and Decision coverage [27] increased exponentially with the coverage level. The gain in fault detection is particularly impressive in the last 10-20 percent coverage. DU coverage performs better than Decision coverage, but is also significantly more expensive as measured by test suite sizes (i.e., number of test cases). Their results are based on seven small programs (141 to 512 LOCs) in which 10 experienced programmers manually seeded faults (130 overall) and for which test pools (of sizes 1,067 to 5,548) were generated to ensure each exercisable coverage unit was covered by at least 30 test cases. Faults that were deemed too easy or too hard to detect (in terms of detection probability) by their test pool were left out of the analysis. Test suites were generated in such a way as to ensure that all coverage and size intervals would be covered by enough randomly generated test suites (e.g., at least 30 test suites per 2 percent coverage interval). Fault detection effectiveness was measured as the proportion of test suites, within each 2 percent coverage interval or two size units, detecting the fault of a faulty program version. The results regarding the relationships between coverage, test suite size, and fault detection effectiveness are not reported in detail as only a couple of graphs are shown for one example program. The paper, however, suggests that this graph is representative of their overall results.

Frankl and Iakounenko [11] also reported very sharp increases in fault detection in the last 10 percent coverage range for All-uses and Decision coverage. The subject program used is the same as in our study and is much larger than the programs used in Hutchins et al. [17] and Frankl and Weiss [13] ($>$ 5,000 NLOC, noncommented lines of code). They reused a randomly generated test pool (10,000 test cases) [26] and based their analysis on 33 actual faults. Out of 33 faulty versions of the program, they only selected those showing, based on the test pool, a failure rate below 0.015, the rationale being that such coverage criteria should be applied after the software has undergone "a fair amount of testing and debugging." Only 11 faulty versions were retained for analysis and, in order to avoid a confounding effect of test suite size, the authors decided, for each version, to generate test suites of fixed sizes (from 20 to 200, depending on the version). Fault detection effectiveness was measured as the percentage of test suites detecting the fault across coverage levels. Large numbers of test suites (10^5 to 10^6) were randomly generated and the authors reported that the number of test suites at higher levels of coverage were, however, fairly small.

To conclude, very few studies report precise experimental results on the cost and effectiveness of control and data flow criteria. Results suggest that whether or not criteria are cost-effective depends on the program and faults. However, all results report that high coverage levels should be used to achieve a high likelihood of fault detection. These results are mostly based on small programs or small numbers of faults, which may in part explain their inconsistency. Results are, in many studies, difficult to interpret because either the effect of test suite sizes is confounded by the effect of coverage levels on fault detection or the size of test suites is fixed in some artificial manner.

A more general issue is related to the creation of test pools. In general, a test pool, also called universe in [11], [12], [13], must be adequate for each considered coverage criterion. This implies that test cases in the pool 1) cover all (most) of what can be covered and 2) form a sample that is representative of the entire input domain to enable the generation of a large variety of adequate test sets. However, for practical reasons, experiments are also typically limited in terms of the size of the test pool they can consider: Each test case must be executed on each faulty version of the subject program and coverage information must be measured.

3 Experimental Description

3.1 Definition of the Experiment

Based on the motivations stated in the introduction, we investigate in this paper seven research questions in the order specified below. Our results must also be compared to existing, reported results and plausible reasons for differences must be investigated. The practical motivations justifying such investigations are also carefully reported.

Q1: Are mutation scores, i.e., proportions of killed mutants, good predictors of actual fault detection ratios? This is important as mutation analysis would allow us to generate large numbers of seeded faults and thus facilitate the statistical analysis of fault detection ratios by decreasing the impact of random variation across test suites.

Q2: What is the cost of achieving given levels of the investigated coverage criteria and how do the criteria compare to each other in this respect? Though we expect P-Use to be more expensive than Decision coverage, which, in turn, should be more expensive than Block coverage, we want to investigate the magnitude of the difference and how the cost of increasing coverage changes along the coverage range.

Q3: Based on mutation analysis, can we determine what levels of coverage, for each criteria, should be achieved to obtain reasonable levels of fault detection effectiveness? Are the last 10-20 coverage percent points important to significantly improving fault detection? This is a question of practical importance as one needs to decide which level of coverage is necessary to achieve predetermined quality criteria.

Q4: Based on mutation analysis, what is the relative cost-effectiveness of the investigated control and data flow coverage criteria? Effectiveness in this context is defined in terms of the fault detection ability of the resulting test suites. There is currently little empirical evidence regarding this matter. This work presents a precise procedure for collecting and analyzing data on this question and reports precise empirical results based on a real system.

Q5: Based on mutation analysis, what is the gain of using coverage criteria compared to random test suites? Given that control and data flow criteria require code analysis that entails significant overhead, are they doing significantly better than random testing? This question is important as practitioners need to determine whether the difference justifies the overhead and there is scant evidence that this is the case.

Q6: Based on mutation analysis, given a coverage criterion, do we still find a statistically significant relationship between coverage level and fault detection effectiveness when we account for test suite size? It could be that test suites that achieve higher coverage are more effective simply because they are bigger. Can we find any evidence for or against this theory? This question is related to Q4 since random test suites should capture the effect of test suite sizes.

Q7: Based on mutation analysis, how is the cost-benefit analysis of coverage criteria affected by variations in fault detection difficulty? For instance, if only very hard-to-find faults are expected to remain in the software under test, how would this affect our assessment of the benefits obtainable by higher coverage levels? Can we calibrate the mutant generation process in order to simulate the effect of different fault detection difficulties? How should such calibration take place? As discussed in Section 4.7, this question was originally motivated by the comparison of our results with the ones of Hutchins et al. [17] as we thought that differences in the overall fault detection probabilities might explain the observed variations.

As presented in Section 4, the main results regarding the above research questions can be summarized as follows:

- In our case study, mutation scores appear to be representative of actual faults found during system testing and operational use (Question Q1). However, as discussed below, such a result is likely to vary across environments and we propose a way to tune the mutation process.
- Results confirm that achieving higher levels of coverage is increasingly more expensive for all criteria (Question Q2). As expected, the cost of achieving a certain coverage level across criteria varies significantly. When criteria are sorted in order of increasing cost, we obtain: Block, C-Use, Decision, P-Use.
- For all criteria, achieving coverage levels close to 100 percent seems to be effective in terms of fault detection (Question Q3). A significant increase in fault detection is still visible in the higher part of the coverage range as relationships between fault detection and coverage appear to be mildly exponential.
- The cost-effectiveness of the investigated coverage criteria appear to be similar (Question Q4). In other words, for a similar test suite size, they seem to detect a similar percentage of faults.
- In terms of cost-effectiveness, there seems to be a practically significant difference between test suites that have been built to achieve coverage objectives and random test suites (Question Q5). The difference in fault detection percentage increases as test suite size increases to converge towards a maximum of 30 percent.
- For all criteria, the fault detection effectiveness of a test suite is not only driven by its size, but also by its coverage (Question Q6). Beyond ensuring a minimum amount of testing, this result confirms that coverage criteria help develop more effective test suites.
- Not only the relationships among fault detection effectiveness, test suite size, and coverage level are affected by fault detection "difficulty," but, in some cases, the shape of the relationship changes significantly (Question Q7). For example, the harder the faults are to detect, the more exponential the relationship between fault detection and coverage level is. In other words, for faults that are highly difficult to detect, achieving very high levels of coverage is even more important. These results are useful in two ways. First, as further discussed in Section 4, they help explain differences between our study and previous ones. Second, they suggest the need to tailor the mutant selection process to the specific difficulty level observed in every development environment.

3.2 Subject Programs

In this paper, we focus exclusively on space.c, the only subject program from our first study [1] that had real faults. This program was originally developed by the European Space Agency, first used in a software engineering study by Pasquini et al. [26], and used subsequently in other experiments. space.c allows the user to describe the configuration of an array of antennas using a specific array definition language (ADL). It reads a text file containing ADL statements, checks its conformance to the ADL grammar as well as specific consistency rules, and performs other computations. It is a 5,905-NLOC C program made of three subsystems (parser, computation, and formatting). (See [33] for further details.) During "testing and operational use" of the program, 33 faults were identified and eliminated and the details of the fault-fixing changes were preserved so that the faults could be selectively reintroduced. Vokolos and Frankl [31] used the program for a study in which they compared the effectiveness of regression test selection strategies. For that study, they generated 10,000 test cases using a randomized input generation tool. These 10,000 test cases formed their experimental "test pool." Rothermel et al. [28] later added enough test cases to ensure that each executable Decision was covered by at least 30 test cases in each direction; this procedure added 3,585 test cases to the pool. The resulting test pool covers 90, 85, 85, and 80 percent of all Blocks, Decisions, C-Uses, and P-Uses present in the program, respectively.[1] The total number of Blocks, Decisions, C-Uses, and P-Uses are 2,995, 1,191, 3,733, and 1,707, respectively.

During the course of their research, Rothermel et al. [28] identified and eliminated five more faults, bringing the total number of versions of the program to 38; however, they found that three of the original versions did not exhibit faulty behaviour, reducing the number of nonequivalent versions to 35. We obtained the program, faulty versions and test pool from the Galileo Research Group Subject Infrastructure Repository at the University of Nebraska -

[1] Though we know the total number of uses in the system, we do not know how many are actually feasible. So, we cannot estimate precisely whether these percentages are close to the proportions of feasible C-Uses and P-Uses. However, all the coverage percentages of the test pool are comparable across criteria.

Lincoln. We compiled and ran the correct (oracle) version and all 38 available versions, recording which test cases caused failures on which faulty versions. After being compiled on our platform (a SunOS 5.8 machine with version 5.8 of the gcc C compiler), we confirmed that three of the original versions did not exhibit any failure for us either. However, one of the versions added by Rothermel et al. also did not exhibit faulty behavior for us. This lowered the number of faulty versions to 34.

3.3 Mutant Generation

To generate mutants of the subject program, we used a mutant generation program first used by Andrews and Zhang [2] to generate mutants for code written in C. To generate mutants from a source file, each line of code was considered in sequence and each of four classes of "mutation operators" was applied whenever possible. Every valid application of a mutation operator to a line of code resulted in another mutant being generated. The four classes of mutation operators were:

- Replace an integer constant C by 0, 1, -1, $((C)+1)$, or $((C)-1)$.
- Replace an arithmetic, relational, logical, bitwise logical, increment/decrement, or arithmetic-assignment operator by another operator from the same class.
- Negate the decision in an `if` or `while` statement.
- Delete a statement.

The first three operator classes were taken from Offutt et al.'s research [23] on identifying a set of "sufficient" mutation operators, i.e., a set S of operators such that test suites that kill mutants formed by S tend to kill mutants formed by a very broad class of operators. They were adapted so that they would work on C programs rather than the Fortran of the original research. The fourth operator, which also appears in [23], was added because the subject program contains a large number of pointer-manipulation and field-assignment statements that would not be vulnerable to any of the sufficient mutation operators.

Some of the resulting mutants did not compile. However, there were so many mutants generated and compiled (11,379) that it was infeasible to run them all on the test pool. We therefore ran the test pool on every 10th mutant generated. Because the number of mutants generated per line did not follow any pattern that would interact with the selection of every 10th mutant (it depends on the constructs on the line only), this constituted a practical procedure for randomly selecting 10 percent of the mutants, taken from a uniform distribution over all the possible mutants. Additionally, this ensured that the whole source code was seeded with faults (and not simply a few functions).

All mutants that were not killed by any test case were deemed to be equivalent to the original program. Though this may not be the case for every mutant, it was thought to be a good enough approximation and it is, in any case, the only option when dealing with large numbers of mutants since automatically identifying equivalent mutants is an undecidable problem [5], [24]. In the end, 736 of these mutants actually exhibited faulty behavior and, so, were used as the analogues of the 34 faulty versions.

3.4 Experimental Design and Analysis

We provide in this section a brief description and justification of the analysis procedure that we used. Additional details will be presented in the next section as we report on the results. We will also compare our experimental design with that of previous work to provide plausible explanations for differences in results.

3.4.1 Experimental Design

We generated a number of random test suites for various coverage criteria and levels, with the aim of obtaining a spread of test suites that span all coverage levels in a balanced manner. These test suites were intended to represent the population of suites that might be built by a test engineer having a specific coverage level goal in mind. Coverage level was measured using the ATAC coverage tool [20].

The algorithm that we used to generate a test suite that achieves at least T percent coverage of a criterion is shown in Fig. 1. Note that a test engineer trying to achieve higher coverage would intelligently select new test cases that target uncovered code and that we simulate this by randomly choosing test cases and discarding them until we find one that covers uncovered code. If we had not discarded test cases that fail to achieve additional coverage, we would have ended up with larger final test suites. These test suites would likely have forced more failures since every additional test case has a chance of forcing a failure, whether or not it covers new code. However, we do not consider that this would simulate a realistic test selection procedure.

Our goal was to find an even spread of test suites from 50 percent coverage to high coverage. Preliminary studies indicated that, for low coverage percentages, the algorithm rarely hit its percentage target precisely, but frequently overshot by one or more percentage points. Therefore, for each of the four coverage criteria, we used this algorithm to randomly generate 10 test suites from the test pool that achieved at least 45 percent (feasible) coverage,[2] at least 46 percent coverage, and so on up to 95 percent coverage. This procedure yielded at least five test suites that achieved between 50.00 percent and 50.99 percent coverage, between 51.00 percent and 51.99 percent coverage, and so on up to 95.00-95.99 percent coverage. We randomly selected five test suites corresponding to each of these intervals (e.g., five that achieved between 50.00 percent and 50.99 percent coverage). We therefore ended up with 230 test suites for each coverage criterion, five for each coverage percentage from 50.00-50.99 to 95.00-95.99. We refer to these test suites as the CS (Coverage Suite) test suites. We did not attempt to generate test suites achieving over 95 percent coverage because it was too difficult to do so for the C-Use and P-Use coverage criteria and would have made our experiment computationally prohibitive.

In order to study research question Q5, we also generated 1,700 random test suites, from the same pool, with each size from one test case to 150 test cases (the size of the largest test suite in the CS). We refer to these as the RS (Random Suite) test suites; they are intended to represent a baseline for comparison to the CS test suites. Indeed, these test suites represent

[2] Coverage values in this paper always refer to *feasible* coverage, which we approximate by the percentage of coverage elements that can be covered by the whole test pool.

```
Algorithm generate_testsuite
Input: Coverage criterion C, target coverage T
Output: Test suite achieving at least T% coverage of criterion C

Set suite = {};
Set suite_coverage = 0;
While (suite_coverage < T) {
   Choose a test case "tc_chosen" randomly from the test pool;
   Add tc_chosen to suite;

   Set new_coverage_value = coverage percentage for suite on criterion C;

   If (new_coverage_value <= suite_coverage) {
       Remove tc_chosen from suite;
   } Else {
       Set suite_coverage = new_coverage_value;
   }
}
Return suite;
```

Fig. 1. Algorithm to generate test suites.

what can be achieved by chance without any specific consideration for coverage. For each of the CS and RS test suites, we then determined which mutants and faults were detected by each test suite and we computed the mutant and fault detection ratios of all test suites.

For each test suite S, the procedure yielded two pieces of summary data: Dm(S), the number of mutants detected by S, and Df(S), the number of faults detected by S. Given the number Nm of nonequivalent mutants and Nf of nonequivalent faults[3] of the subject program, we calculated the *mutation detection ratio* Am(S) of each test suite S as Dm(S)/Nm and the *fault detection ratio* Af(S) as Df(S)/Nf.

3.4.2 Experimental Analysis

To address our first research question (Q1), we need to determine whether Af and Am differ for various coverage levels and coverage criteria. We will not only look at their difference but also at the Magnitude of Relative Error (MRE) [10], defined as $|\mathrm{Af} - \mathrm{Am}|/\mathrm{Af}$. MRE is a common measure for evaluating the accuracy of predictive systems and our research question can be reexpressed as a prediction problem: Can we accurately predict Af based on Am? A positive answer would require the MRE to be small enough to be negligible from a practical standpoint. From a statistical standpoint, we will also test whether the difference between Af and Am is statistically significant, but it is important to realize here that our goal is to assess whether such a difference, whether statistically significant or not, would be of practical significance[4] [19]: Would that affect any decision regarding the application of any of the investigated coverage criteria? We will also look at how MRE varies according to coverage levels in order to determine whether we can provide a consistent answer regardless of the targeted level of coverage. In addition, we will determine whether a linear regression model can be used to predict Af from Am and what error intervals we can expect.

We investigate question Q2 by analyzing and modeling the relationships between the coverage level of all four criteria and test suite size for the CS test suites, using appropriate nonlinear regression analysis.[5] It is assumed that the effort associated with a coverage level is proportional to test suite size, that is, the number of test cases drawn from the test pool.

Question Q3 can be addressed by analyzing the relationships between Am and coverage level, focusing on assessing the significance of change in Am for the higher part of the coverage level range.

To address research question Q4, we analyze and model the relationships of Am with test suite size for the CS test suites, once again using nonlinear regression analysis. Am is a measure of the benefit of achieving a certain level of coverage, whereas test suite size is our measure of cost. Using and comparing the modeled Am/Test suite Size relationships, we can then compare the cost-benefit or cost-effectiveness of using various coverage criteria at different coverage levels.

To investigate question Q5, we model and fit the relationship between Am and test suite size for the RS test suites and compare this relationship with the fitted criteria relationships based on the CS suites. Indeed, we can assume that, for all coverage criteria and random test suites, test suite size is proportional to the cost of running and verifying test cases. However, the cost of generating random test cases is clearly much less than that of test cases based on control and data flow criteria. Therefore, for a given test suite size, only a significant difference in fault detection would warrant the use of control and data flow criteria. How much of a difference is required is, of course, context-dependent (e.g., analysis tools, testers' expertise).

Q6 is addressed by simply performing bivariate regression considering both test suite size and coverage level as covariates to explain variations in Am. If both covariates turn out to be statistically significant, we can then conclude that they both contribute to drive fault detection and that

[3] Similar to the notion of nonequivalent mutant, a nonequivalent fault is a fault that can be detected by at least one test case in the test pool.

[4] Statistical significance is concerned with whether a research result is due to chance or sampling variability; practical significance is concerned with whether the result is useful in the real world.

[5] We actually linearize the relationships between the dependent and independent variables through appropriate transformations. This allows us to use linear least-square regression and facilitates the analysis and the interpretation of results.

TABLE 1
Descriptive Statistics for Af-Am

	Mean	Median	2.5%	97.5%	Min	Max	Std Dev
Block	0.013	0.011	-0.092	0.139	-0.14	0.16	0.057
C-Use	0.018	0.012	-0.078	0.131	-0.14	0.16	0.057
Decision	0.016	0.023	-0.104	0.139	-0.22	0.23	0.065
P-Use	0.015	0.014	-0.104	0.132	-0.154	0.227	0.063

the effect of test suites on fault detection is not just the result of their size.

We address question Q7 by modeling and comparing the relationships between Am, test suite size, and coverage level for subsets of seeded mutants that are relatively more difficult to detect given our test pool. In order to compare our results with existing work, we focus on two subsets of mutants that are detected by less than 5 percent and 1.5 percent of test cases in the test pool, respectively. These percentages are comparable to those reported in existing studies and we then try to explain the differences between our results and those reported based on the impact of variations in fault detection probabilities on the relationships between Am, test suites size, and coverage level. The practical implications of our results on the calibration of the mutation analysis process is then considered.

4 Analysis Results

In this section, we present the results of our analysis, addressing each research question in turn.

4.1 Are Mutation Scores Good Predictors of Actual Fault Detection Rates?

This analysis addresses research question Q1, the question concerning whether the effectiveness of a coverage-goal test suite on mutants was a good predictor of its effectiveness on faults. To study this question on a per-test-suite level, we calculated the difference $(\mathrm{Af} - \mathrm{Am})$ for each test suite. Descriptive statistics for $(\mathrm{Af} - \mathrm{Am})$ are presented in Table 1. First, we can observe from the mean (positive but close to zero) and the quantiles (min, max, 2.5 percent, 97.5 percent) that we have rather symmetric and unbiased distributions for all coverage criteria. When performing a matched pairs t-test of the difference between Af and Am for Block coverage, it is actually significant $(\mathrm{p} = 0.0003)$, but the mean difference (Mean) is not of practical significance as a difference slightly above 1 percent will not likely have a practical impact. Given that similar results were obtained on the other three coverage criteria (statistically significant differences—$\mathrm{p} < 0.001$—of little practical significance), we can therefore conclude that Am will not significantly and systematically underestimate or overestimate Af.

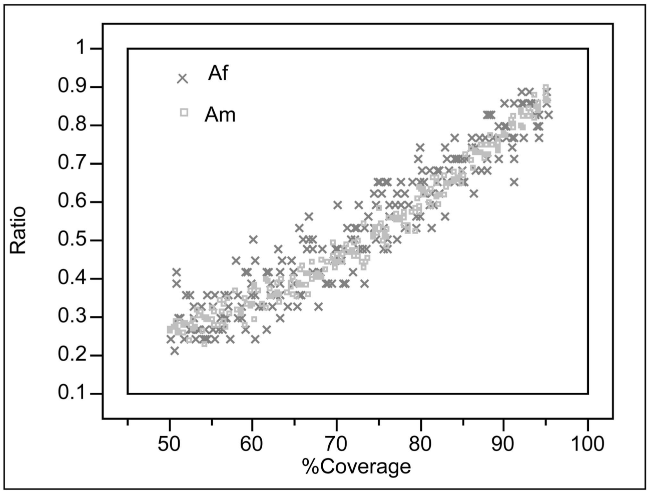


Fig. 2. Observations for Am/Af ratios versus Block coverage.

Fig. 2 shows another interesting angle on the results. It is a scatterplot of all the observations showing the relationship between Am and Af, respectively, and the Block coverage level (%Coverage). We can see that Am and Af have a very similar relationship with %Coverage, but the variance for Af observations is much larger (i.e., they are more spread out). This is to be expected as Am is a ratio based on a large number of mutants, whereas Af is based on 34 faults only and is therefore much more subject to random variations due to the random selection of test cases across test suites. We obtained similar results for the other three criteria, thus confirming this observation is not specific to Block coverage.

More importantly, we should now look at the MRE distribution, which normalizes $|\mathrm{Af} - \mathrm{Am}|$ by Af and thus expresses prediction errors as a percentage of the actual value to predict. Table 2 shows the descriptive statistics for the distribution of MRE values across all test suites, for all four coverage criteria. We can see that the average MRE ranges from 0.083 to 0.098, that is, a relative error under 10 percent. We also see that, in 95 percent of the cases, MRE can be expected to be under 31 percent for Block and 25 percent for the others.

If we now look at the relationship between MRE and the achieved coverage level, we see that, for all coverage criteria, the average MRE tends to decrease as the percentage of coverage increases. A representative example is shown in Fig. 3, where a smoothing spline[6] [9] of the average MRE is drawn for Block coverage. We can clearly see the downward trend by looking at the spline and the decrease in variance of the observations along the Y-axis as the percentage of coverage increases. For coverage levels above 85 percent, the MRE will in nearly all cases be below 15 percent. From a practical perspective, it is difficult to imagine how such a low MRE could practically affect

[6] The smoothing spline is an excellent way to get an idea of the shape of the expected value of the distribution of MRE across the percentage of coverage achieved. The cubic spline method uses a set of third-degree polynomials spliced together such that the resulting curve is continuous and smooth at the splices (knot points).

TABLE 2
Descriptive Statistics for MRE

	Mean	Median	2.5%	97.5%	Min	Max	Std Dev
Block	0.098	0.084	0.005	0.311	0	0.418	0.077
C-Use	0.083	0.066	0.002	0.253	0	0.432	0.072
Decision	0.088	0.075	0.007	0.253	0	0.414	0.065
P-Use	0.083	0.065	0.002	0.249	0	0.286	0.064

decision making regarding the selection of a coverage criterion or a coverage level; a testing decision based on an experiment with mutants would be unlikely to be cast into doubt over such a low MRE. This is an important result as, in practice, the objective will be to achieve high coverage levels and, as a result, MRE will be even lower and show less variability across test suites, thus making the prediction of Af based on Am more accurate.

Another way to look at the association between Af and Am is to perform a linear regression between the two. Fig. 4 shows the computed regression line for Block coverage and the 95 percent confidence intervals for the predictions. It has a R^2 of 0.908 and a slope close to 1 (0.967), as we would expect if Am were a good, unbiased predictor of Af. Based on such a linear regression model, we can therefore see that an unbiased and accurate prediction of Af based on Am is possible. Other coverage criteria show nearly identical results, with a R^2 ranging from 0.83 to 0.90.

We conclude that the effectiveness of the CS test suites in finding real faults in the space.c program can be accurately predicted by the effectiveness of those suites in finding mutants generated from space.c. This finding is consistent with the results of [1], which found that the same predicted relationship holds for test suites of given sizes chosen randomly from the same entire test pool.

Because the above section showed that Am is a good predictor of Af, we use the large number of mutants at our disposal to facilitate the analysis addressing questions Q2 to Q7: The subsequent subsections use Am as a measure of fault detection effectiveness. Using the large number of mutants will lead to a reduced random variation in the fault detection effectiveness evaluation of test suites (as visible in Fig. 2 for example) and will therefore facilitate the analysis of relationships between fault detection, coverage, and test suite size.

4.2 Comparing the Cost of Coverage Criteria

We address question Q2 in this section. When looking at the relationships between coverage level and size, we can observe that achieving higher levels of coverage becomes increasingly expensive. This data can be accurately fitted with an exponential regression model: $\mathrm{Coverage} = \mathrm{a\ Size}^{\mathrm{b}}$, where a and b are coefficients to be estimated through linear regression.[7] This model is intuitive as we expect size to be zero when there is no coverage and coverage might not increase proportionally to size. Fig. 5 shows an example of fitted curve for Block coverage. The relationships among coverage criteria differ significantly, as illustrated by the variations among coefficients given in Table 3. We also provide the coefficients' 95 percent confidence intervals (CI) —to make it clear that the differences are not due to chance—and the overall fit of the model (R^2).[8] The varying cost of achieving a certain level of coverage across criteria is clearly visible when displaying the relationships of coverage percentage to test suite size: The fitted curves are shown for all four coverage criteria in Fig. 6. Following the recommendation in [16], we also compute the surface areas under the curves (in the size interval 0-100). This provides a way to quantify the differences among curves (i.e., the cost of achieving coverage) in a way which is independent from any particular size value. Table 3 reports both the absolute surface areas and the relative surface areas (in parentheses) when normalized by the smallest area (P-Use). We see that P-Use does not differ much from Decision but the differences with Block and C-Use are, however, significant.

As expected, a given level of coverage is easier to achieve for Block and then C-Use, Decision, and P-Use, in that order. Since C-Use has no subsumption relationship with Decision or P-Use, it depends entirely on the program whether the test suites are larger for a given coverage level. However, P-Use coverage subsumes Decision coverage and should therefore be at least as difficult to achieve, though a clear difference is only visible toward higher size values.

The small difference between the P-Uses and Decision curves may be due to the test pool we use. Recall that, though Rothermel et al. [28] ensured that feasible Decisions were exhaustively covered, we have no guarantee that the test pool is covering all feasible P-Uses. We know that 80 percent and 85 percent of the P-Uses and C-Uses are covered, respectively, but is that close to the actual percentage of feasible coverage? In comparison, 85 percent of Decisions are covered by the same test pool and perhaps this is an indication that the actual percentage of feasible P-Uses is higher but that the test pool falls short of achieving maximum coverage. Though the difference is not large, if the test pool had covered a higher percentage of P-Uses, we might observe a more pronounced difference between Decision and P-Use. If we assume that the actual feasible P-Use coverage is comparable to Decision or C-Use, that is 85 percent, we would then obtain the curves in Fig. 7 with the same test suites. Note that the P-Use curves conform better to the relationship we would expect between Decision and P-Use: A specific level of coverage is more expensive to achieve for P-Use than Decision. The surface area under the new P-Use curve is 7,012, which is

[7] The model is linearized as $\ln(\mathrm{Coverage}) = \ln(\mathrm{a}) + \mathrm{b}\ \ln(\mathrm{Size})$. In our fitted coverage regression models, b is less than 1 as achieving higher levels of coverage is increasingly more expensive. Note that such a model is flexible as it can fit a decreasing $(\mathrm{b} < 1)$ or increasing $(\mathrm{b} > 1)$ slope for Coverage as test suite size increases.

[8] Though our objective in this paper is not to build prediction models, we show coefficients of determination (R^2) to demonstrate that the curves and their underlying models closely fit our data. We can thus compare these curves knowing they are really representative of the data.

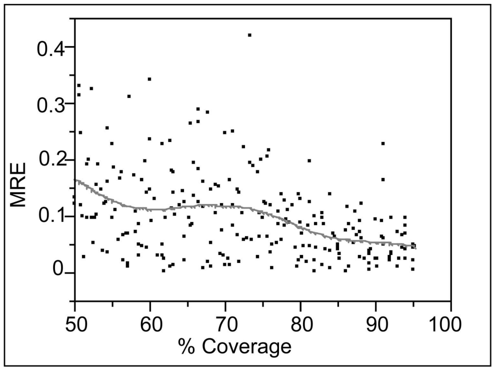


Fig. 3. Block average MRE as a function of coverage level.

significantly lower than the previous value (7,450) and that of Decision (7,455).

4.3 Are There Minimal, Required Levels of Coverage?

In the previous section, we studied the cost of coverage criteria with research question Q2. In this section, we focus on the effectiveness with research question Q3. Fig. 8 shows the observations, fitted relationship, and 95 percent confidence intervals between Am and the coverage level for Block coverage $(\mathrm{R}^2 = 0.98)$. Similar analyses were performed for the other three criteria where a similar form of relationship and R^2 were obtained. All regression lines are displayed in Fig. 9 and show what we expect: A given level of coverage kills more mutants for P-Use and Decision (they show no significant difference), C-Use, and Block, in that order. We can see that curves range from being nearly linear (Decision) to mildly exponential (Block).

As discussed in Section 4.2, the small difference between P-Use and Decision may be due to the fact that the current test pool covers a lower percentage of feasible P-Uses than Decisions. If the actual feasible coverage percentage for P-Use were 85 percent (as for C-Use and Decision), then we would obtain the models and curves in Fig. 10 with the same test suites. As for the analysis in Section 4.2, the difference between Decision and P-Use then becomes visible and, as we would expect given the subsumption relationship between the two criteria, at a given coverage level, P-Use detects more faults than Decision. Though we cannot be absolutely sure, this new result for P-Use is more plausible. The construction of the test pool and its impact on the results will be further discussed in Section 4.8.

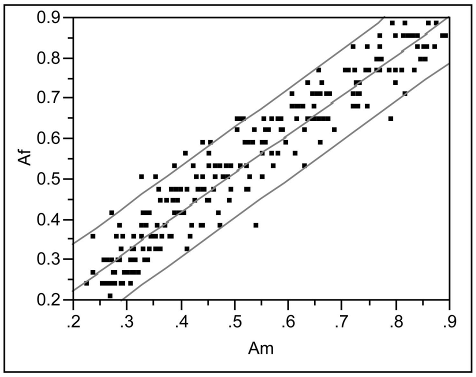


Fig. 4. Block regression line between Af and Am.

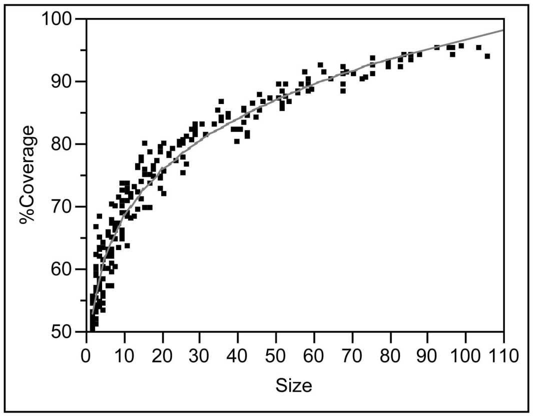


Fig. 5. Relationship between Block coverage level (percent) and test suite size.

Several past studies investigated that issue with great care. Hutchins et al. [17], based on their reported diagram, found strong exponential relationships between fault detection effectiveness and the coverage levels for Decision and DU coverage where the last 10-20 coverage percents detect an increasingly large percentage of faults. For example, only a DU coverage above 90 percent can lead to a fault detection effectiveness above 50 percent! Frankl and Iakounenko [11] reported results along the same lines but even more extreme, where, for all eight faulty versions considered, the probability of fault detection is dismally low and sharply increases after 90 percent for Decision and All-Uses coverage.

The results obtained in our study are not nearly as extreme as what was reported in the studies discussed above. The curves show relationships that are at best mildly exponential. However, what is consistent is that high levels of coverage are required to achieve decent fault detection rates. Attempting to achieve a coverage level close to 100 percent is definitely yielding significant additional fault detection. The difference between our results and those reported will be further discussed in Section 4.7.

4.4 Comparing the Cost-Effectiveness of Coverage Criteria

Question Q4 is addressed in this section. Using the mutants that we generated, we investigate the cost-effectiveness of coverage criteria by analyzing the relationships between the percentage of faults exposed (effectiveness) and the size of test suites (our surrogate measure for cost). Fig. 11 shows the observations (test suites) and modeled relationship between Am and test suite size for the CS test suites using Block coverage. We modeled the relationship as an exponential relationship, as in the previous section. It fit the data well $(\mathrm{R}^2 = 0.976)$ and is a rather intuitive model: An empty test suite should detect 0 percent of the faults and faults get increasingly harder to detect as test cases are added to the test suite $(\mathrm{b} < 1)$. As further discussed below, the relationships for the other three coverage criteria are very similar.

It is worthwhile noting two caveats here. First, size is an imperfect measure of cost for coverage-oriented test suites; some coverage measures involve expensive analyses and

TABLE 3
Regression Coefficients, Their 95 Percent CI, and R^2 for All Coverage (Percent)-Size Models

Coverage-Size Model	a	95% CI	b	95% CI	R^2	Area under curve (0-100)
Block	47.58	Lower: 46.92 Upper: 48.25	0.154	Lower: 0.149 Upper:0.159	0.977	8379 (1.125)
C-Use	40.97	Lower: 40.12 Upper: 41.83	0.178	Lower: 0.179 Upper:0.184	0.943	7894 (1.059)
Decision	33.91	Lower: 33.20 Upper: 34.63	0.213	Lower: 0.207 Upper:0.219	0.966	7455 (1.0007)
P-Use	37.10	Lower: 36.06 Upper: 38.16	0.189	Lower: 0.1809 Upper:0.1966	0.972	7450 (1)

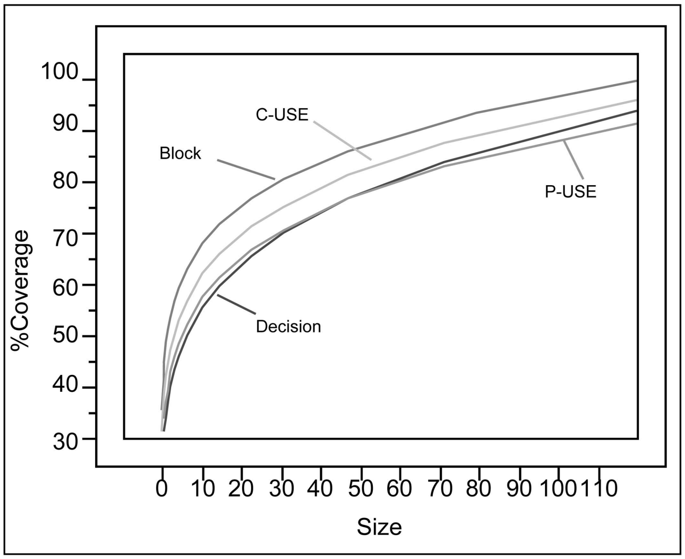


Fig. 6. Comparing coverage level-test suite size relationships.

the last few test cases added to achieve a given percentage of coverage may be considerably more difficult for a tester to find than the preceding ones. Second, in practical settings, we are often concerned with three-way trade-offs among size, coverage, and fault detection effectiveness (represented here by Am), but, in order to simplify and clarify the presentation, we present here only graphs showing pairs of these three measures. Hence, in this section and the next, we present graphs of test suite size compared to Am, which must be interpreted in light of the knowledge that the data points represent test suites constructed with particular coverage goals in mind.

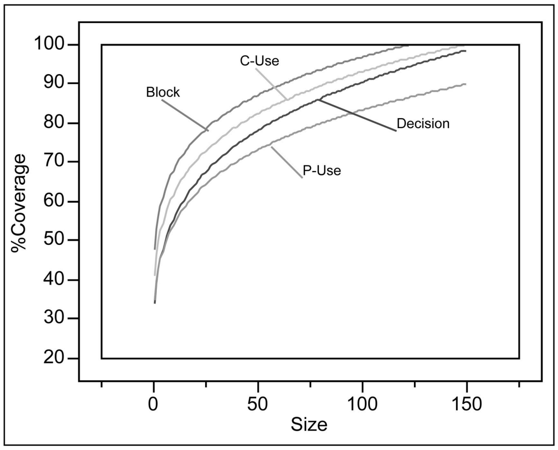


Fig. 7. The impact of an actual feasible P-use coverage of 85 percent.

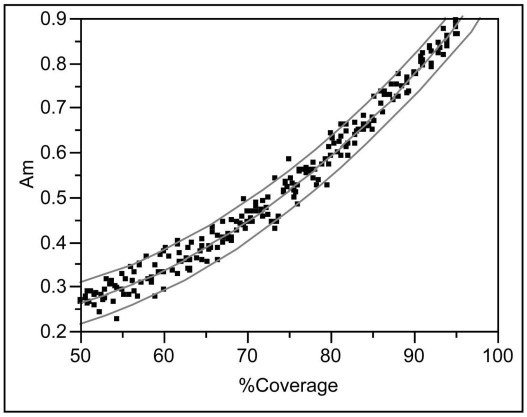


Fig. 8. Am-coverage level relationship for block.

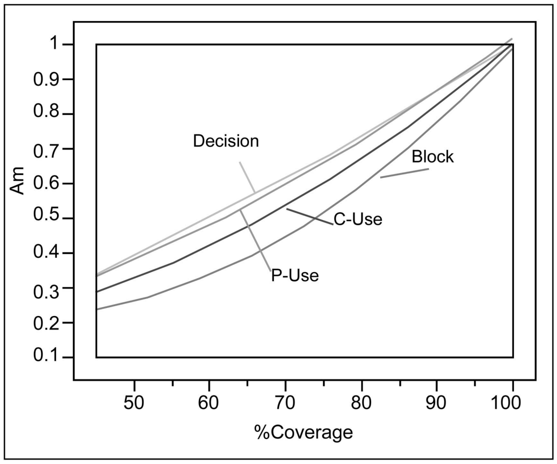


Fig. 9. Comparing Am-coverage level relationships across criteria.

Table 4 reports on all the regression coefficients and R^2 values for all four coverage criteria (all coefficients are significant at $p < 0.0001$). It also shows the surface areas under the curves [16]. The regression coefficients, goodness of fit, and surface areas of the models are all very similar. What this implies is that none of the four criteria is more cost-effective than the others. For a given test suite size, they would all find a very similar number of faults. However, the more demanding criteria would force the tester to generate larger test suites to achieve a specific coverage level. For example, the median test suite sizes for the last coverage interval (95-95.99 percent) are 97, 109, 126, and 135 for Block, C-Use, Decision, and P-Use, respectively. To summarize, the various coverage criteria entail a varying testing intensity for a given coverage level, but none of them is more cost-effective than the others at uncovering faults.

It is useful to compare these results to the ones of Hutchins et al. [17]. Although they did not provide detailed data or fitted models, from the graphics they report for one of the faulty programs considered, we can tell that their Fault detection-Size relationship looked mostly linear for Decision (Edge) coverage and DU (All-Uses) coverage shows perhaps a small plateau for the highest size intervals. Recall, however, that their faults were manually seeded, filtered out when detected by too few or too many test cases (below 3 or above 350), and were small in number (from 7 to 39 faults per program). This difference will be further investigated in Section 4.7 by controlling for fault detection probability (relative to our test pool) and assessing its impact on the Am-Test suite size relationship. Furthermore the programs considered were substantially smaller and the measure of fault detection effectiveness for each faulty version was defined as the percentage of test suites in each size interval (of width 2) that contained at least one test case detecting the fault. None of the other reported studies focused on the relationship between fault detection and test suite size as a way to compare the cost-effectiveness of criteria.

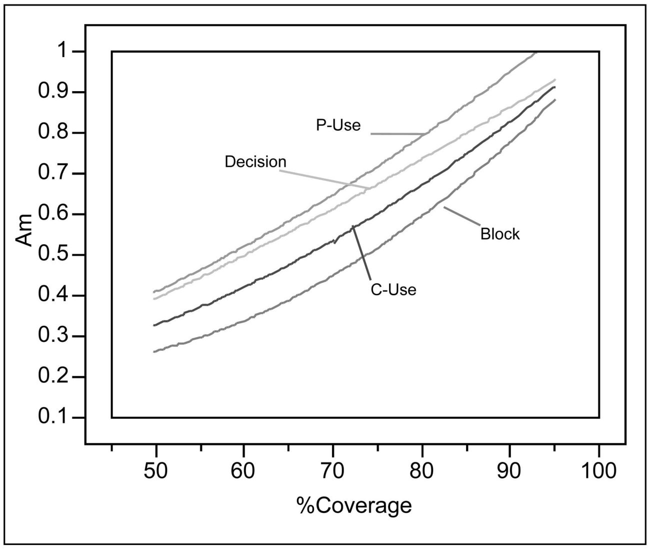


Fig. 10. The effect of an actual 85 percent feasible coverage for P-use.

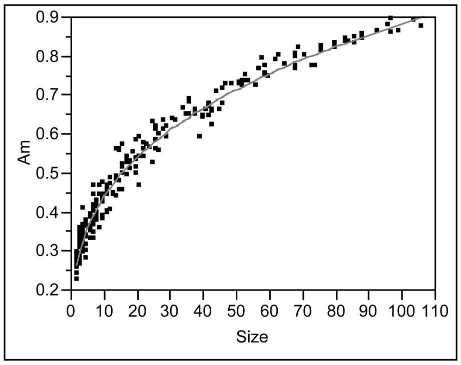


Fig. 11. Relationship between Am and test suite size for Block coverage.

4.5 Comparing Coverage Criteria to Random Test Suites

This section addresses question Q5. Using coverage criteria and, in particular, data flow criteria requires substantial code analysis and the generation of test cases to achieve a certain coverage level is not a fully automated activity [14]. In order for such criteria to be useful in practice, they should perform much better than randomly generated test suites of identical size, as discussed in Section 3.4.2. To investigate differences between random test suites and coverage criteria, we used the RS test suites described in Section 3.4. We fitted the relationship between Am and test suite size based on the available data as for the coverage criteria Am curves, except that a smoothing spline was used [9] to achieve better fit ($R^2 = 0.96$) of the data as no simple exponential or any other standard model would fit well (Fig. 12). The fitted model therefore represents the cost-effectiveness that can be achieved simply by chance and

TABLE 4
Regression Coefficients, R^2, and Surfaces Areas for All Am-Size Models

Am-Size (Model)	a	b	R^2	Area under curve (0-100)
Block	0.213	0.309	0.976	67.52
C-use	0.219	0.300	0.977	67.66
Decision	0.226	0.292	0.973	67.12
P-use	0.226	0.287	0.972	65.85

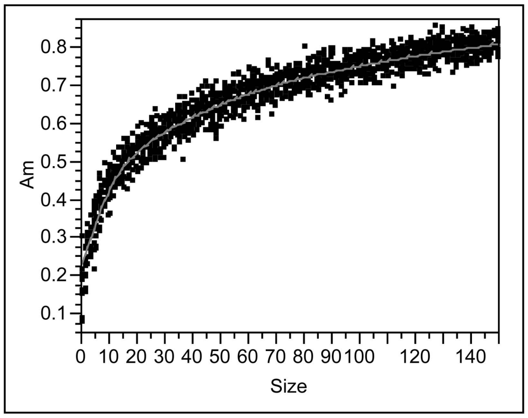


Fig. 12. Am-size relationship for random testing.

models the effect of having increasingly larger test suites on fault detection.

Fig. 13 depicts the relationship between Am and test suite size for both the CS and RS test suites. We can clearly see that the random curve is below the coverage criteria curves, with an increasing difference when test suite size increases. Though no difference was identified among the four coverage criteria, there is a clear difference with random test suites: Coverage criteria test suites are clearly more cost-effective and more so for higher coverage levels and larger test suites. Such a result therefore suggests that control and dataflow criteria can provide useful guidance and can perform significantly better in terms of fault detection than generating large numbers of random test cases. The question remains whether the observed difference justifies the overhead of using control or data flow criteria.

One issue that could be raised is whether the test pool we use is suitable for generating random test suites. The reason is that the original test pool generated by Vokolos and Frankl [31] was completed by Rothermel et al. [28] to achieve better decision coverage and has therefore not been entirely randomly generated. This augmentation may possibly bias the results by making the random test suites look more effective than they actually are. To ensure that the augmentation did not affect our conclusions, we reran our analysis with the original test pool of 10,000 test cases.

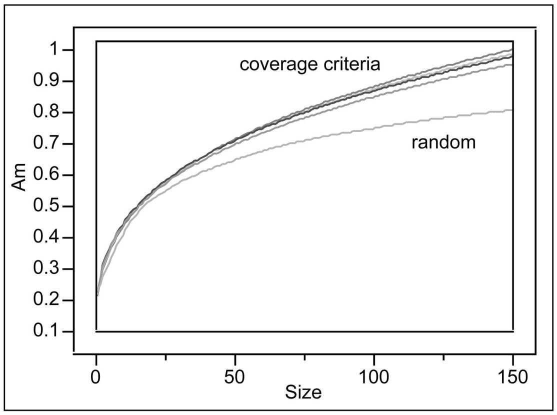


Fig. 13. Comparing coverage criteria cost-effectiveness with random testing.

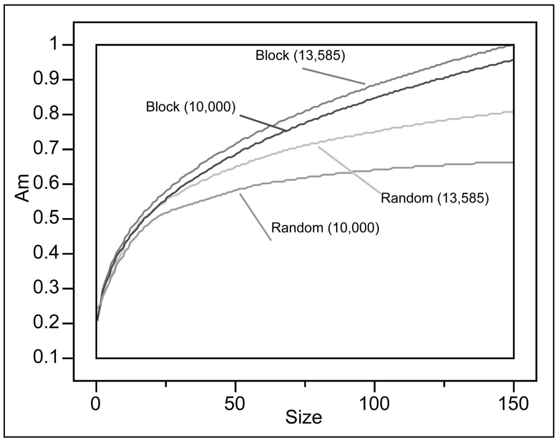


Fig. 14. Am for random test suites with the Vokolos and Frankl test pool for Block.

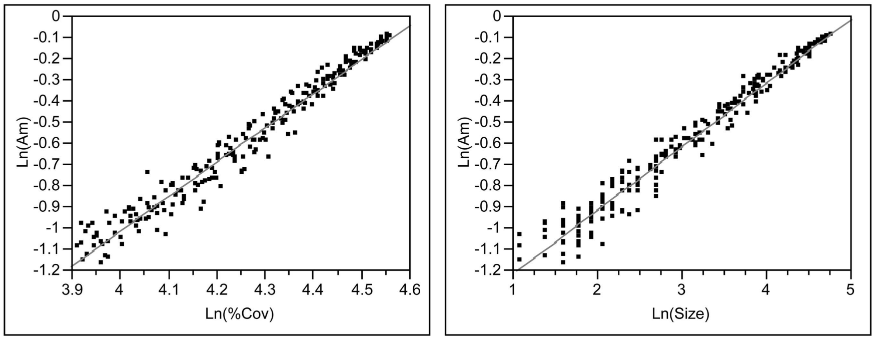
The results were very similar in terms of the shapes of the curves, except that the difference between random test suites and the coverage criteria was indeed larger, as shown in Fig. 14 for Block coverage. For example, for a test suite size of 100, the Am differences for the Block and random test suites are 0.13 and 0.20 for the augmented and original test pools, respectively. Similar, larger differences were also observed for the other three criteria. Such differences, however, do not strongly affect our conclusions.

Table 5 reports the areas below the Am-Size curves for all four criteria, normalized by the random test suite areas obtained with the Vokolos and Frankl test pool (10,000) and the complete test pool (13,585), respectively. We can clearly see, in quantitative terms, that the random test suites are significantly less cost-effective than CS test suites based on the former, but the difference decreases significantly on the latter, thus confirming, for all criteria, what is visible in Fig. 14 for Block.

As mentioned above (Section 2.2), the work by Frankl and Weiss [13] reported inconsistent results in terms of the difference between the Decision and All-Uses coverage criteria and random test suites (null criterion). This difference with our results may be due to the much smaller size of their programs or the fact that they based their results on a single fault per program, but it is difficult to tell for certain. Hutchins et al. [17] reported much stronger results in terms of the superiority of DU coverage and Decision coverage versus random test suites of the same size in the upper coverage level range (> 91 percent). On average, DU coverage showed an increase in fault detection

TABLE 5
Criteria Areas Normalized by Random Testing Areas

Am-Size (Model)	Normalized area under curve (0-100)
Block	1.217, 1.093
C-Use	1.219, 1.095
Decision	1.209, 1.086
P-Use	1.186, 1.061



(a) (b)

Fig. 15. Univariate regression for Am versus (a) C-use coverage level and (b) test suite size.

between 1 percent and 68 percent, depending on the coverage level considered. Within the same coverage level range, Decision showed an increase ranging from 40 percent to 75 percent compared to random test suites of the same size. From Fig. 13, we can see that, as in Hutchins et al. [17], the gain of using coverage criteria grows as test suite size increases (for higher coverage levels). However, the absolute percentage of increase in mutant detection grows to a maximum of approximately 20 percent for the higher coverage levels.

4.6 Is the Effect of Coverage Simply a Size Effect?

This section addresses research question Q6. It is in many ways related to question Q5 since, if differences in fault detection effectiveness were only due to test suite size, random test suites would perform as well as coverage criteria test suites. Recall that the study performed by Frankl and Weiss [12] on small programs showed that the difference between the fault detection effectiveness of All-Uses and Decision versus random test suites was not always significant. Similarly to their logistic regression analysis, we can perform here a multivariate regression analysis using both test suite size and coverage level (all four criteria) as model covariates. Because of the form of the relationships between Am and these two variables, we will run the regression analysis on their log-transformed versions. As visible from the univariate regressions for C-Use coverage in Fig. 15, we obtain (nearly) linear relationships.

When running a regression analysis using these two variables as covariates, both turn out to be statistically significant $(p < 0.0001)$ and show a positive regression coefficient, thus both contributing to increasing Am as they increase. The regression model shows a $R^2 = 0.984$ and details about parameter estimates are shown in Table 6.

TABLE 6
Multiple Regression Results for C-Use Coverage (%Cov) and Test Suite Size

Term	Estimate	Std Error	t Ratio	Prob>|t|
Intercept	-4.519131	0.19496	-23.18	<.0001
Ln(%Cov)	0.8080354	0.052461	15.40	<.0001
Ln(Size)	0.1566097	0.009689	16.16	<.0001

The fit in the nontransformed space shows an $R^2 = 0.99$ and Fig. 16 shows the relationship between Am and the predictions we obtain using the model in Table 6. It is therefore clear that, when taking into account test suite size and C-Use coverage, we explain most of the variance in Am. This suggests that both size and coverage play a complementary role in explaining fault detection, despite their strong correlation ($R^2 = 0.93$ for C-Use).[9] In other words, C-Use coverage seems to have an impact on fault detection that extends beyond just a size effect. As summarized in Table 7, we obtained similar results for the other three coverage criteria where both size and coverage are significant with $p < 0.0001$. In addition to regression coefficients, Table 7 also reports the standard errors of coefficient estimates and the R^2 in the log-transformed space and the original space.

4.7 Impact of Fault Detection Probabilities

This section investigates question Q7. We have seen in Section 4.4 that Am can be accurately modeled as having an exponential relationship with test suite size, showing an increasing difficulty in detecting more faults as size increases (exponent coefficient $b < 1$). However, how is that relationship affected by the detection difficulty of faults? Answering this question is useful in two ways: 1) It may help explain some of the differences with other studies and 2) it may have practical implications in terms of calibrating mutation analysis to real faults in a specific context.

To investigate this question, we focus on two subsets of mutants: "hard" and "very hard" mutants that are detected by less than 5 percent and 1.5 percent of the test cases in the test pool, respectively. We then calculate Am only with respect to the number of mutants in each of these categories. Fig. 17 shows three sets of observations for the Block coverage test suites, corresponding to the two subsets of mutants plus the entire set. Note that observations are connected by line segments to better render the shape of the relationships in the figure. We can see that both subsets

[9] We verified, by looking at the Variance Inflation Factor (VIF) [29], that collinearity between the two covariates was not an issue for the parameter estimates.

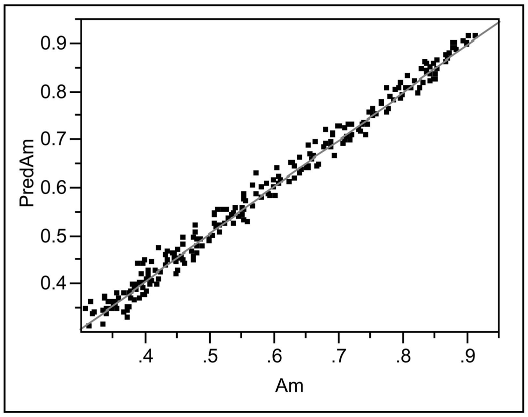


Fig. 16. Prediction of Am (PredAm) using the model in Table 6.

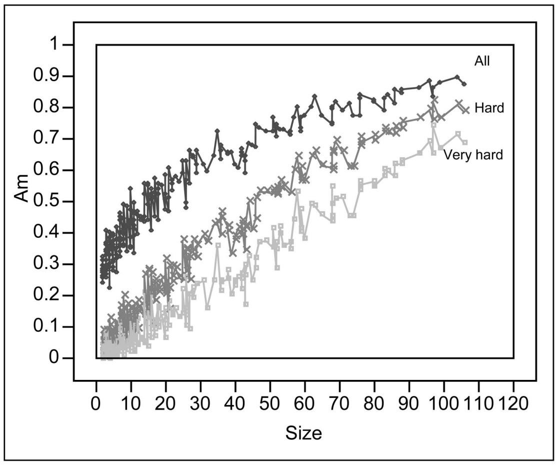


Fig. 17. Am-size observations for all, "hard," and "very hard" mutants (Block).

show a nearly linear relationship, which sharply differs from what was observed in Section 4.4 (based on all observations). Similar relationships can be observed for the remaining three coverage criteria. Interestingly, Hutchins et al. [17] also reported nearly linear relationships between fault detection and test suite size and we have seen in Section 4.4 that our result was visibly different. They reported that 113 seeded faults were removed from their analysis because they were deemed too easy to detect (a minimum of 350 test cases, between 6 percent and 33 percent of the pool test cases would find them). The average detection probability across our 736 nonequivalent mutants is 16 percent and many of our mutants would have been discarded if we had followed the procedure by Hutchins et al. Therefore, this new analysis suggests that the difference between Hutchins et al. [17] and our results is due to the fact that the faults they had seeded were more difficult to detect.

If we now turn our attention to the Am-Coverage relationship, Hutchins et al. [17] had reported an exponential relationship showing an increasing fault detection rate as the achieved coverage percentage was increasing $(b > 1)$. Frankl and Iakounenko [11] showed even more extreme results with sharp increases in fault detection in the last 10-20 percent coverage level range (Decision, All-Uses). Again, as visible in Fig. 18, if we analyze "hard" and "very hard" mutants, we obtain sharply different relationships for Block Coverage than those observed in Section 4.3 (based on all observations). (Similar results were obtained for other criteria.) For "hard" and "very hard" mutants, the relationship looks very much like the ones reported in [11], [17], respectively. Recall that, as mentioned above, Hutchins et al. discarded the faulty versions they considered too easy to detect. A similar procedure, but even more extreme, was adopted in [11], where only faulty versions showing a failure rate (based on the test pool) of less than 1.5 percent were considered. We therefore get a quite consistent picture where we obtain exponential relationships between fault detection rates and coverage level, with sharp increases in the highest 10-20 percent coverage levels, when only faults with low detection probabilities are considered. However, recall that we have shown in Section 4.1 that our set of mutants is representative of actual faults. We therefore believe that the cost-effectiveness results we report in Section 4.4 are more realistic than the data for the "hard" and "very hard" curves in Fig. 18.

What the above analysis also tells us is that the relationships we have modeled between Am and test suite size (Size) or achieved coverage level (%Coverage) are very sensitive to the detection difficulty of the faults considered, as captured by the detection probability by the pool of all test cases. In practice, the detection difficulty of faults depends on characteristics of the system under test and the development process that was used. In order to use mutation analysis (in the sense of Offutt [25]) to assess whether a test suite is effective in a specific context (system and organization), it may be necessary to filter out a subset of "easy" mutants among all generated mutants, so as to

TABLE 7
Summary of Multiple Regression Results with the Four Coverage Criteria

	Ln(Size) (p<.0001)	Std Error	Ln(%Cov) (p<.0001)	Std Error	R^2 (Ln)	R^2
Block	0.14	0.01	1.04	0.08	0.98	0.99
Decision	0.11	0.01	0.85	0.06	0.98	0.99
C-Use	0.16	0.009	0.81	0.05	0.98	0.99
P-Use	0.18	0.01	0.54	0.05	0.97	0.98

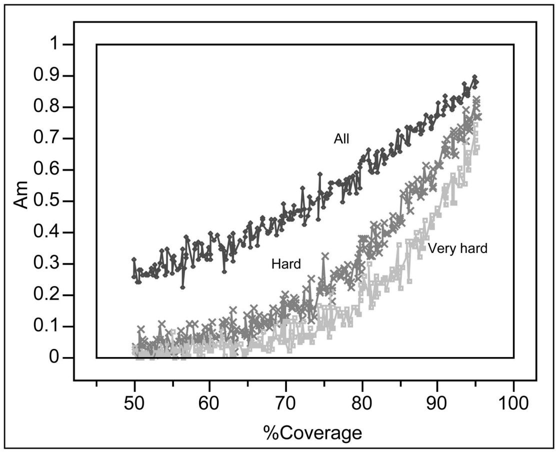


Fig. 18. Am-coverage (percent) observations for all, "hard," and "very hard" mutants (Block).

obtain Am-Size relationships that are representative of real faults. The question is then what probability threshold to use in a specific context? A practical approach would be to take a sample of faults on a completed project, like the Space program, and perform an analysis similar to ours to determine what threshold allows Am to predict Af in the most accurate way possible. Such a calibration process would allow the determination of an optimal threshold, where generated mutants would systematically be discarded from the detection effectiveness analysis when more than a certain percentage of test cases in a project test suite would kill them. The assumption underlying this calibration process is that such a threshold would be stable within an application domain and when a stable development process is used so that it could be determined on a project and used on future projects during the testing phase.

4.8 Threats to Validity

No data is perfect and no analysis yields 100 percent trustable results. It is, however, important to identify the threats to validity of an experiment and carefully assess the likelihood of such threats and their potential consequences. This section discusses different types of threats to the validity of our experiments: *internal*, *external*, and *construct validity* [6], [32].

One issue related to internal validity is due to the fact that we reused, without any modification, a program that has been widely used in previous experiments. The program, test pool, and faults we are using were not selected based on any particular criteria, except that they were well-prepared and historically important. However, the randomly generated test pool is two orders of magnitude larger than our test suites and should be large enough to generate realistic, random variation in the construction of test suites and cover a large proportion of feasible Decisions and Def-Use pairs.

After an initial random generation of 10,000 test cases, the test pool that we used was completed (with 3,585 test cases) with the intent of covering all executable Decisions, as reported in Section 3.2, but no attempt was made to explicitly cover all C-Uses and P-Uses. This augmentation may have resulted in artificially inflating the coverage levels observed for these two criteria, which are computed using what can be considered feasible coverage based on the test pool. Indeed, the test pool coverage for P-Uses and C-Uses can be different from the actual feasible coverage, thus affecting our comparisons of the cost of coverage criteria as well as their relative fault detection effectiveness. However, the test pool is rather large and covers a large proportion of C-Uses (85 percent) and P-Uses (80 percent) in the program, proportions which are not very different from the proportions of Blocks (90 percent) and Decisions (85 percent) covered.

It is expected that the results of our study would vary depending on the mutation operators selected; the ones we used, as discussed in Section 3.3, were selected based on the literature available so as to be a minimal but sufficient set. The selection of every 10th mutant may also represent a threat to internal validity; we mitigated this threat by confirming that this selection did not interact with any pattern of mutation operation application and is thus a practical and valid way to randomly select mutants.

Another issue is related to the way the test suites are built. Would a human tester generate more cost-effective structural test suites? Would this result in a larger difference between random test suites and the ones based on structural criteria? It is, of course, only possible to answer such a question by running experiments involving human subjects. Such experiments, however, present a number of challenges as, for example, it is difficult to imagine how large numbers of test suites could be generated within a reasonable period of time. But, it is probably the case that the difference we observe in our study between random and criteria-based test suites is close to what a human tester would observe.

External validity relates to our ability to generalize the results of the experiment to industrial practice. Since only one program with real faults was used (albeit a program that has been studied extensively in the past), it will be very important to replicate this study on other subject programs where real faults have been identified. Though the program is small by the usual industrial standards, it is much larger than many of the small programs used in previous, related empirical studies. Ideally, for running experiments such as the one in this paper, one would want to use the largest test pool possible, but doing so leads to prohibitive execution times when executing a large number of mutants on a large number of program versions while collecting control and data flow information. It is also expected that results would probably vary, depending on the specific development process, as it would determine the fault characteristics (detectability, type) to be expected at different verification and validation phases. We have seen, for example, that results are very sensitive to the detection probability of faults and a procedure to tune the mutation analysis process to a set of representative faults was proposed to address this issue. However, even in situations where Am would not be a good predictor of Af, the important question is whether the relative ranking of the effectiveness of two test methods using Am would be a good predictor of their relative ranking using Af. If it is, it would lead to the conclusion that it is a safe experimental practice to automatically generate mutants from source code and to compare the effectiveness of test methods by comparing how many mutants the test methods kill. More analyses such as the one presented in this paper are required to answer this question with a high level of confidence.

Construct validity concerns the way we defined our measurement and whether it measures the properties we really intend to capture: the fault detection effectiveness of test sets, the detectability of faults, and the cost of testing. Our justifications for detection effectiveness and detectability were given at length above when discussing the size of test sets and their construction. We model testing cost by the size of test suites; this measure may fail to capture all dimensions of testing cost when considering different coverage criteria since it may be more difficult to find test cases that achieve some coverage criteria than others. For example, data flow criteria might be more difficult to use as they involve data flow analysis. This entails that they should only be used if they make a practically significant difference justifying the additional cost.

5 Conclusion

This paper reports on an empirical study performed on one industrial program with known system testing faults. We investigate the feasibility of using mutation analysis to assess the cost-effectiveness of common control and data flow criteria. Our results show that mutation analysis is potentially useful to assess and compare test suites and criteria in terms of their cost-effectiveness. In other words, in our context, it has been shown to yield results that are similar to what would be obtained with actual faults. If this result is confirmed, it suggests that mutation analysis can be used in a research context to compare and assess new testing techniques. But, it is also applicable in a more practical context where a development organization must empirically determine what levels of coverage to achieve to attain acceptable detection rates.

Among the control and data flow criteria studied here, none is more cost-effective than the other, though more demanding criteria (e.g., C-Use and P-Use) entail larger test suites that detect more faults. In other words, their relationships between fault detection and test suite size are similar. However, as expected, their cost (in terms of test suite size) varies significantly for a given coverage level.

There is a sizable difference between the cost-effectiveness of random test suites and that of coverage criteria, thus justifying the use of the latter. But, it is unclear whether the observed differences always justify the additional overhead related to control and data flow analysis and the identification of covering test cases. Further data analysis also shows that the impact of coverage criteria on fault detection effectiveness is not only due to an increase in test suite size, thus explaining the cost-effectiveness difference between random test suites and coverage criteria.

The probability of detection of faults given a test pool (also referred to as a test universe) strongly affects the shape of relationships between fault detection, coverage levels, and test suite sizes. This fact seems to explain the differences across existing studies and this one. Since we have shown that the seeded mutants were representative of real faults on our subject program, whereas the faults considered in other studies were much harder to detect, our results seem to be more realistic. However, should such fault detection probabilities differ widely across development environments, it would have a strong impact on the testing requirements in terms of coverage levels in order to attain acceptable fault detection rates.

Because empirical studies of control and data flow criteria are still rare, there are still numerous open questions to investigate. For example, it is very important for practitioners to figure out ways to decide on appropriate coverage levels given a required fault detection rate. Techniques for them to perform trade-offs are required. We cannot converge toward adequate solutions to these problems if we do not gain a better understanding of the cost-effectiveness of control and data flow criteria and the factors that affect their applicability. As for mutation analysis, we need to devise procedures to tailor it to a specific environment. This paper proposes an initial strategy based on discarding mutants using their detection probability by a test suite, but this approach needs to be further investigated and evaluated from a practical standpoint.

Acknowledgments

Many thanks to Gregg Rothermel and Hyunsook Do for valuable discussions and suggestions and for sending the authors the space.c program and all the associated artifacts. Thanks also to all the researchers who worked on and improved the space.c subject program and artifacts over the years. Lionel Briand's work was partly supported by a Canada Research Chair (CRC) grant. Jamie Andrews, Lionel Briand, and Yvan Labiche were further supported by NSERC Discovery grants. Akbar Siami Namin was supported by an Ontario College of Graduate Studies (OCGS) graduate scholarship.

References

[1] J.H. Andrews, L.C. Briand, and Y. Labiche, "Is Mutation an Appropriate Tool for Testing Experiments?" *Proc. IEEE Int'l Conf. Software Eng.*, pp. 402-411, 2005.

[2] J.H. Andrews and Y. Zhang, "General Test Result Checking with Log File Analysis," *IEEE Trans. Software Eng.*, vol. 29, no. 7, pp. 634-648, July 2003.

[3] B. Beizer, *Software Testing Techniques,* second ed. Van Nostrand Reinhold, 1990.

[4] L.C. Briand, Y. Labiche, and Y. Wang, "Using Simulation to Empirically Investigate Test Coverage Criteria," *Proc. IEEE/ACM Int'l Conf. Software Eng.*, pp. 86-95, 2004.

[5] T.A. Budd and D. Angluin, "Two Notions of Correctness and Their Relation to Testing," *Acta Informatica,* vol. 18, no. 1, pp. 31-45, 1982.

[6] D.T. Campbell and J.C. Stanley, *Experimental and Quasi-Experimental Designs for Research.* Houghton Mifflin Company, 1990.

[7] W. Chen, R.H. Untch, G. Rothermel, S. Elbaum, and J. von Ronne, "Can Fault-Exposure-Potential Estimates Improve the Fault Detection Abilities of Test Suites?" *Software Testing, Verification, and Reliability,* vol. 12, no. 4, pp. 197-218, 2002.

[8] R.A. DeMillo, R.J. Lipton, and F.G. Sayward, "Hints on Test Data Selection: Help for the Practicing Programmer," *Computer,* vol. 11, no. 4, pp. 34-41, Apr. 1978.

[9] R.L. Eubank, *Spline Smoothing and Nonparametric Regression.* Marcel Dekker, 1988.

[10] N.E. Fenton and S.L. Pfleeger, *Software Metrics: A Rigorous and Practical Approach,* second ed. PWS Publishing, 1998.

[11] P.G. Frankl, O. Iakounenko, "Further Empirical Studies of Test Effectiveness," *Proc. Sixth ACM SIGSOFT Int'l Symp. Foundations of Software Eng.*, pp. 153-162, Nov. 1998.

[12] P.G. Frankl and S.N. Weiss, "An Experimental Comparison of the Effectiveness of the All-Uses and All-Edges Adequacy Criteria," *Proc. Fourth Symp. Testing, Analysis, and Verification*, pp. 154-164, 1991.

[13] P.G. Frankl and S.N. Weiss, "An Experimental Comparison of the Effectiveness of Branch Testing and Data Flow Testing," *IEEE Trans. Software Eng.*, vol. 19, no. 8, pp. 774-787, Aug. 1993.

[14] D. Hamlet and J. Maybee, *The Engineering of Software.* Addison Wesley, 2001.

[15] R.G. Hamlet, "Testing Programs with the Aid of a Compiler," *IEEE Trans. Software Eng.*, vol. 3, no. 4, pp. 279-290, 1977.

[16] M. Harder, J. Mellen, and M.D. Ernst, "Improving Test Suites via Operational Abstraction," *Proc. 25th Int'l Conf. Software Eng.*, pp. 60-71, May 2003.

[17] M. Hutchins, H. Froster, T. Goradia, and T. Ostrand, "Experiments on the Effectiveness of Dataflow- and Controlflow-Based Test Adequacy Criteria," *Proc. 16th IEEE Int'l Conf. Software Eng.*, pp. 191-200, May 1994.

[18] S. Kim, J.A. Clark, and J.A. McDermid, "Investigating the Effectiveness of Object-Oriented Testing Strategies with the Mutation Method," *Software Testing, Verification, and Reliability*, vol. 11, no. 3, pp. 207-225, 2001.

[19] R.E. Kirk, "Practical Significance: A Concept Whose Time Has Come," *Educational and Psychological Measurement*, vol. 56, no. 5, pp. 746-759, 1996.

[20] M.R. Lyu, J.R. Horgan, and S. London, "A Coverage Analysis Tool for the Effectiveness of Software Testing," *IEEE Trans. Reliability*, vol. 43, no. 4, pp. 527-535, 1994.

[21] A.M. Memon, I. Banerjee, and A. Nagarajan, "What Test Oracle Should I Use for Effective GUI Testing?" *Proc. IEEE Int'l Conf. Automated Software Eng. (ASE '03)*, pp. 164-173, Oct. 2003.

[22] A.J. Offutt, "Investigations of the Software Testing Coupling Effect," *ACM Trans. Software Eng. and Methodology*, vol. 1, no. 1, pp. 3-18, 1992.

[23] A.J. Offutt, A. Lee, G. Rothermel, R.H. Untch, and C. Zapf, "An Experimental Determination of Sufficient Mutation Operators," *ACM Trans. Software Eng. and Methodology*, vol. 5, no. 2, pp. 99-118, 1996.

[24] A.J. Offutt and J. Pan, "Detecting Equivalent Mutants and the Feasible Path Problem," *Software Testing, Verification, and Reliability*, vol. 7, no. 3, pp. 165-192, 1997.

[25] A.J. Offutt and R.H. Untch, "Mutation 2000: Uniting the Orthogonal," *Proc. Mutation*, pp. 45-55, Oct. 2000.

[26] A. Pasquini, A. Crespo, and P. Matrelle, "Sensitivity of Reliability-Growth Models to Operational Profiles Errors vs Testing Accuracy," *IEEE Trans. Reliability*, vol. 45, no. 4, pp. 531-540, 1996.

[27] S. Rapps and E.J. Weyuker, "Selecting Software Test Data Using Data Flow Information," *IEEE Trans. Software Eng.*, vol. 11, no. 4, pp. 367-375, Apr. 1985.

[28] G. Rothermel, R.H. Untch, C. Chu, and M.J. Harrold, "Prioritizing Test Cases for Regression Testing," *IEEE Trans. Software Eng.*, vol. 27, no. 10, pp. 929-948, Oct. 2001.

[29] T.P. Ryan, *Modern Regression Methods.* Wiley, 1996.

[30] P. Thévenod-Fosse, H. Waeselynck, and Y. Crouzet, "An Experimental Study on Software Structural Testing: Deterministic versus Random Input Generation," *Proc. 21st Int'l Symp. Fault-Tolerant Computing*, pp. 410-417, June 1991.

[31] F.I. Vokolos and P.G. Frankl, "Empirical Evaluation of the Textual Differencing Regression Testing Technique," *Proc. IEEE Int'l Conf. Software Maintenance*, pp. 44-53, Mar. 1998.

[32] C. Wohlin, P. Runeson, M. Host, M.C. Ohlsson, B. Regnell, and A. Wesslen, *Experimentation in Software Engineering—An Introduction.* Kluwer, 2000.

[33] W.E. Wong, J.R. Horgan, A.P. Mathur, and A. Pasquini, "Test Set Size Minimization and Fault Detection Effectiveness: A Case Study in a Space Application," Technical Report TR-173-P, Software Eng. Research Center (SERC), 1997.

James H. Andrews received the BSc and MSc degrees in computer science from the University of British Columbia, and the PhD degree in computer science from the University of Edinburgh. He worked from 1982 to 1984 at Bell-Northern Research, from 1991 to 1995 at Simon Fraser University, and from 1996 to 1997 on the FormalWare project at the University of British Columbia, Hughes Aircraft Canada, and MacDonald Dettwiler. He is currently an associate professor in the Department of Computer Science at the University of Western Ontario, in London, Canada, where he has been since 1997. His research interests include software testing, semantics of programming languages, and formal specification. He is a member of the IEEE and the IEEE Computer Society.

Lionel C. Briand received the PhD degree in computer science, with high honors, from the University of Paris XI, France. He is currently a visiting professor at the Simula Research Laboratories, Oslo, Norway. He is on sabbatical leave from the Department of Systems and Computer Engineering, Carleton University, Ottawa, Canada, where he is a full professor and has been granted the Canada Research Chair in Software Quality Engineering. Before that, he was the software quality engineering department head at the Fraunhofer Institute for Experimental Software Engineering, Germany. He also worked as a research scientist for the Software Engineering Laboratory, a consortium of the NASA Goddard Space Flight Center, CSC, and the University of Maryland. He has been on the program, steering, or organization committees of many international, IEEE and ACM conferences. He is the co-editor-in-chief of *Empirical Software Engineering* (Springer) and is a member of the editorial board of *Systems and Software Modeling* (Springer). He was on the board of the *IEEE Transactions on Software Engineering* from 2000 to 2004. His research interests include model-driven development, testing and quality assurance, and empirical software engineering. He is a senior member of the IEEE and a member of the IEEE Computer Society.

Yvan Labiche received the BSc degree in computer system engineering from the Graduate School of Engineering: CUST (Centre Universitaire des Science et Techniques, Clermont-Ferrand), France. He received the master's degree in fundamental computer science and production systems in 1995 (Université Blaise Pascal, Clermont-Ferrand, France). While completing the PhD degree in software engineering, received in 2000 from LAAS/CNRS in Toulouse, France, he worked with Aerospatiale Matra Airbus (now EADS Airbus) on the definition of testing strategies for safety-critical, on-board software, developed using object-oriented technologies. In January 2001, he joined the Department of Systems and Computer Engineering at Carleton University, Ottawa, Canada as an assistant professor. His research interests include: object-oriented analysis and design, software testing in the context of object-oriented development, and technology evaluation. He is a member of the IEEE.

Akbar Siami Namin received the BS degree in computer science from the University of Kerman, Iran, and the MS degree in computer science from Lakehead University, Canada. He worked from 1993 to 2001 in industry and the IT field. In 2003, he joined the National Research Council of Canada, London, Ontario, as a guest worker. He is currently a PhD student in the Department of Computer Science at the University of Western Ontario, London, Canada. His research interests are software engineering and testing, distributed computing, and security.

▷ **For more information on this or any other computing topic, please visit our Digital Library at** www.computer.org/publications/dlib.