4-1-2000

An Empirical Study of the Effects of Incorporating Fault Exposure Potential Estimates into a Test Data Adequacy Criterion

Wei Chen
Oregon State University

Gregg Rothermel
Oregon State University, gerother@ncsu.edu

Roland H. Untch
Middle Tennessee State University

Jeffery von Ronne
University of California – Irvine

University of Nebraska-Lincoln, Computer Science and Engineering
Technical Report # TR-UNL-CSE-2000-0004; issued 04/01/2000

Technical Report 00-60-04, Computer Science Department, Oregon State University, March, 2000.

An Empirical Study of the Effects of Incorporating Fault Exposure Potential Estimates into a Test Data Adequacy Criterion

Wei Chen
Department of Computer Science
Oregon State University
Corvallis, OR
chenwei@cs.orst.edu

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

Roland H. Untch
Computer Science Department
Middle Tennessee State University
Murfreesboro, TN
untch@mtsu.edu

Jeffery von Ronne
Department of Information and Computer Science
University of California – Irvine
Irvine, CA
jronne@ics.uci.edu

ABSTRACT
Code-coverage-based test data adequacy criteria typically treat all code components as equal. In practice, however, the probability that a test case can expose a fault in a code component varies: some faults are more easily revealed than others. Thus, researchers have suggested that if we could estimate the probability that a fault in a code component will cause a failure, we could use this estimate to determine the number of executions of a component that are required to achieve a certain level of confidence in that component's correctness. This estimate, in turn, could be used to improve the fault-detection effectiveness of test suites. Although this suggestion is intriguing, no empirical studies have directly examined it. We therefore conducted an experiment to investigate the effects of incorporating an estimate of fault exposure probability into the statement coverage test data adequacy criterion. The results highlight several cost-benefits tradeoffs with respect to the incorporation of the estimate.

1. INTRODUCTION
Test data adequacy criteria are measures used to evaluate whether a set of test data is sufficient, and guide testers in the generation of test cases. Code-coverage-based test data adequacy criteria measure adequacy in terms of coverage of source code components, such as statements, decisions, or definition-use interactions, requiring that each component be exercised by at least one test case. These criteria have been the subject of a great deal of research and experimentation (e.g. [1; 9; 14]).

Code-coverage-based test data adequacy criteria typically treat all code components as equal: they assume that one test case exercising each component is sufficient. In practice this assumption is unrealistic. The probability that a test case can expose a fault in a code component varies with several factors, including whether the test case executes the component, whether it causes the fault to create a change in program state, and whether it causes that change in state to propagate to output [4; 5; 11; 15; 17; 18].

Several researchers have therefore conjectured that if we could estimate the probability that a fault in a code component will cause a failure, we could use this estimate to improve the fault-detection effectiveness of code-coverage-based testing [4; 5; 7; 16; 18]. For example, an estimate of the probability that a fault in a component will cause a failure could be coupled with an overall "confidence" requirement to estimate the number of executions of the component that are necessary to achieve a certain probability of that component's correctness [5; 7; 18].

This suggestion is intriguing; however, in our search of the research literature, we could discover no empirical studies that have directly assessed it. Voas [18] reports results of a study assessing the correlation between PIE (propagation, infection, and execution) analysis sensitivity estimates and failures observed in random testing. Goradia [4] reports results of a study in which an estimate of fault propagation probability is assessed for correlation with actual fault exposure data. Neither of these studies, however, examined the effects of directly incorporating such estimates into test data adequacy criteria.

If the conjecture that fault exposure probability estimates could be used to improve the fault-detection effectiveness of code-coverage-based testing could be supported, this would motivate further research on cost-effective techniques for obtaining such estimates, and on techniques for incorporating such estimates into testing. If successful, such research could help testers distribute testing resources more effectively and improve the quality of testing.

We therefore conducted an experiment to investigate the effects of incorporating an estimate of fault exposure probability (which we refer to as a *fault exposure potential estimate*) into the statement coverage test data adequacy criterion. Our results indicate that the incorporation of such an estimate into that criterion can improve the fault-detection effectiveness of test suites that meet the criterion; however, the effects of incorporating the estimate vary with the program under test, the nature of the faults contained in the program, and the level of confidence required of the testing. Our results highlight several interesting cost-benefits tradeoffs with respect to the incorporation of the estimate.

2. PRELIMINARIES

2.1 Fault Exposure and Test Adequacy

The related notions that some faults are more easily exposed than others and that some source code components are more easily tested than others have been frequently addressed in the research literature. Several researchers (see e.g. [4; 5; 7; 11; 15; 17; 18]) have proposed or investigated models of various aspects of fault-detection phenomena. These models in general express the probability that a test case can expose a fault in a code component, if that component contains a fault, as a combination of three factors: (1) whether the test case executes the component, (2) whether it causes the fault to create a change in program state, and (3) whether it causes that change in state to propagate to output.[1]

Estimates of these factors can be combined to estimate the probability that a fault in a code component will cause a failure under a particular input distribution [18]. Following suggestions by Hamlet [7] and Voas [18], we can use such an estimate to determine the number of test cases that are needed to obtain a certain level of confidence in the correctness of a code component, as follows. Let x be a code component, let p_l be the estimated probability that a fault in x will cause a failure, and let c be the confidence that the failure probablity of x is less than p_l. In this case, the number of test cases hn that must be executed through x to obtain confidence level c is given by the equation:

$$hn = \frac{ln(1-c)}{ln(1-p_l)} \quad (1)$$

For practical purposes two special cases involving equation (1) should be considered. First, p_l may be estimated as 0 or 1, in which case the value of hn is undefined. In this case, a prudent choice for hn (since p_l is an estimate) is 1. Second, for values of p_l between 0 and 1, hn may have a fractional value. In this case hn may be a non-integer and, to retain the required level of confidence, must be rounded up.

The application of equation (1) (with the two adaptions just outlined) to a set of code components at a given level of confidence defines a set of *hit numbers*, one for each component. These hit numbers specify the number of executions of each component that are necessary to achieve the required confidence in the correctness of that component. A code-coverage-based test data adequacy criterion incorporating estimates of the probability that a fault in a code component will cause a failure can be realized by requiring that each component be exercised by a number of test cases equal to or exceeding its hit number. In theory, such a criterion could be defined in terms of various types of code components, including statements, decisions, or data dependencies, provided that (1) coverage of that component can be measured, (2) the notion of what it means for such a component to contain a fault can be defined, and (3) appropriate estimates of the probability that a fault in that component will cause a failure can be obtained. In this work, we focus on the use of individual program statements as components, due to the relative simplicity of that approach and the availability of tools and estimates that operate at that level.

[1]A related issue involves the probability that a component contains a fault (e.g. [10]); we do not address that issue.

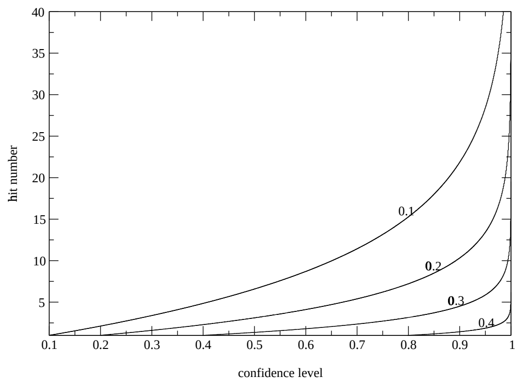

Figure 1: Hit number versus confidence level for four fault exposure probabilities.

To provide a sense of the requirements of such a test adequacy criterion, Figure 1 depicts the relationship among fault exposure probability estimates, confidence levels, and hit numbers. The figure shows, for four fault exposure probability estimates (0.1, 0.2, 0.4, 0.8) the hit numbers required to achieve various confidence levels. The figure indicates that for a given fault exposure probability estimate, as confidence level increases, hit number increases, and that the rate of increase accelerates. In other words, at high levels of confidence, obtaining an increase in confidence level requires a much larger boost in hit numbers than is required to obtain the same increase in confidence level at low levels of confidence. The figure also shows that when fault exposure probability is low, the hit number required to achieve high confidence is much larger than when fault exposure probability is high.

2.2 Estimating Fault Exposing Potential

Having addressed the issues in defining code-coverage-based test data adequacy criteria that incorporate estimates of the probability that a fault will cause a failure, we now consider the calculation of those estimates.

Voas [18] provides one method for performing such calculations, in the form of *PIE* (propagation, infection, and execution) analysis. PIE analysis assesses the probability that, under a given input distribution, if a fault exists in code component x, it will result in a failure. This probability, termed the *sensitivity* of x, is estimated by combining independent estimates of three probabilities: (1) the probability that x is executed (*execution probability*), (2) the probability that a change in x can cause a change in program state (*infection probability*), and (3) the probability that a change in state propagates to output (*propagation probability*). PIE analysis uses various methods to obtain these estimates: (1) simple code instrumentation to estimate execution probability; (2) a variant of *weak mutation* [8] in which syntactic changes are applied to x and then the state after x is examined for affects to estimate infection probability; and (3) state perturbation, in which the data state following x is altered and then program output is examined for differences to estimate propagation probability.

Voas suggests that sensitivity estimates could be used in Equation 1 to calculate the number of executions of a component that are required to obtain a certain confidence in that component's correctness. For the purpose of calculating hit numbers, however, this approach has two disadvantages.

First, by factoring in execution probabilities, sensitivity measures the probability that a fault will cause a failure relative to an input distribution. In code-coverage-based testing, however, we are interested in the probability that, *if a test case executes* a code component x containing a fault, that fault will propagate to output. It is possible for x to have very high [low] infection and propagation probabilities with respect to the inputs that execute it, even though it has a very low [high] execution probability relative to an input distribution. The incorporation of execution probabilities into sensitivity estimates thus distorts the measure of the likelihood that a given test case that reaches x will expose a fault in x. For code-coverage-based testing, a more appropriate measure would consider only infection and propagation.

A second drawback of sensitivity in this context involves its treatment of propagation and infection estimates. Sensitivity analysis separately calculates these estimates, and uses a conservative approach to combine them. This approach can overpredict the probability that an arbitrary input will expose a fault, and result in low estimates of that probability. Such low estimates, utilized to determine hit numbers by the process discussed in the preceding section, could yield excessively high hit numbers.

Thus, in this work, we adopt a different estimate of the probability that a fault in code component x will cause a failure. As in sensitivity analysis, we use mutation analysis [3; 6] to create m mutations of x. We then execute the program on a universe of test inputs, and determine, for each test case t_i that executes x, the number n_i of mutants exposed by that test case. Suppose that there are k test cases that execute x, and together, the sum of the n_i $(1 \leq i \leq k)$ for the k test cases equals n_s. In this case the mutation analysis process has caused x to be executed $k \times m$ times, We use this value $(k \times m)$ to divide n_s, obtaining an average value that indicates, for each test case t_i that executes x, the probability that t_i will reveal a mutant of x. We call the resulting value the *fault exposing potential* (FEP) estimate for x, described more formally by the following equation:

$$FEP_x = \frac{\sum_{i=1}^{k} n_i}{m \times k} \qquad (2)$$

We call a test suite created, by the process outlined in the preceding section to satisfy hit numbers calculated using FEP estimates a *fault-exposing-potential-coverage-adequate* (FEPC-adequate) test suite.

An issue in implementing the foregoing process involves the handling of *equivalent mutants*: mutants that cannot be exposed by any input to the program. In principal, we should eliminate such mutants from consideration, because they do not represent exposable faults; however, in practice, this is a difficult task. Thus, another approach is to make no attempt to distinguish the equivalent mutants, instead, treating all mutants as faults that could potentially be exposed. FEP estimates gathered by this approach are underestimates of the FEP estimates that would be calculated given knowledge of mutant equivalence. In our experimentation we deal with over 150,000 mutants, and equivalent mutant identification is not feasible; thus, we take this second approach.

The results of our empirical study are undoubtedly influenced by our adoption of this particular estimate of the probability that a fault will cause a failure, and in a narrow context, our results reflect the efficacy of this estimate. In a wider context, however, it is important to note that other estimates do or could exist. Goradia [4] presents one such approach, in which impact graphs are analyzed to estimate propagation probabilities. Alternatively, one could adopt Voas' sensitivity estimate, despite the drawbacks we have suggested of it. Another approach might make use of constrained mutation [12], which utilizes a subset of the full set of mutation operators. Our primary interest in this work is not the particular estimate that we adopt, but the question whether given such an estimate, we could use it to create test suites that are more effective than those created by simple code coverage criteria.

3. THE EXPERIMENT

The research questions we wished to investigate can be informally stated as follows:

RQ1: Can incorporation of FEP estimates into a statement-coverage test data adequacy criterion improve the fault-detection effectiveness of test suites?

RQ2: How does the fault-detection effectiveness of FEPC-adequate test suites change as confidence changes?

RQ3: How does the size of FEPC-adequate test suites change as confidence level changes?

RQ4: Do differences in programs affect the fault-detection effectiveness of FEPC-adequate test suites?

RQ5: Do differences in faults affect the fault-detection effectiveness of FEPC-adequate test suites?

3.1 Measures

To address our research questions we require measures of the fault-detection effectiveness of a test suite and of test suite size. To measure test suite size, we focus simply on the number of test cases in the test suite.

Measuring fault-detection effectiveness is not quite as simple. Given a program and a fault set for that program, we define the fault-detection effectiveness of a test suite for that program as the percentage of faults in the fault set that can be detected by that test suite. We refer to this measure of a test suite's effectiveness as the test suite's *efficacy*. More formally, given program P and fault set F for P, where F contains $|F|$ faults, and given test suite T, if the execution of T on P reveals $|F_r|$ of the faults in F, the efficacy of T for P and F is given by $\frac{|F_r|}{|F|} * 100\%$.

This method of measuring fault-detection effectiveness calculates effectiveness relative to a fixed set of faults. This approach also assumes that faults have equal importance, an assumption that typically does not hold in practice. The approach also does not differentiate between test suites that detect faults multiple times (i.e. more than one test case in the test suite detects the fault) and test suites that detect a fault a single time.

Program	LOCs	Mutant Pool Size	Test Pool Size	Fault Pool Size
print_tokens	402	4030	4130	7
print_tokens2	483	4346	4115	10
replace	516	9622	5542	32
schedule	299	2153	2650	9
schedule2	297	2828	2710	10
tcas	138	2876	1608	41
tot_info	346	5898	1052	23
space	6218	132163	13585	38

Table 1: Experiment subjects.

3.2 Experiment Instrumentation

3.2.1 Programs

We used eight C programs as subjects (see Table 1). The first seven programs were collected initially by researchers at Siemens corporation for use in experiments with dataflow and control-flow based test adequacy criteria [9]; we call them the *Siemens programs.* The Siemens programs perform a variety of tasks: `tcas` is an aircraft collision avoidance system, `schedule` and `schedule2` are priority schedulers, `tot_info` computes statistics given input data, `print_tokens` and `print_tokens2` are lexical analyzers, and `replace` performs pattern matching and substitution. The eighth program, `space`, is an interpreter for an array definition language (ADL) used within a large aerospace application.

3.2.2 Test pool and test history

For each of the seven Siemens programs the Siemens researchers created a *test pools* of black-box test cases using the *category partition method* and the Siemens Test Specification Language tool [13]. They then augmented this set with manually created white-box test cases to ensure that each exercisable statement, edge, and definition-use pair in the base program or its control flow graph was exercised by at least 30 test cases. This process produced test pools of the sizes shown in Table 1.

`Space` has a test pool of 13,585 test cases. The first 10,000 test cases were randomly generated by Vokolos and Frankl [19], the remaining test cases were added by authors of this study so that most executable branches in the program[2] were exercised by at least 30 test cases.

For our experiment, we considered each program P with test pool U, we recorded, for each test case in U and each statement in P, whether not that statement was exercised by that test case. This information was used to create individual test suites for the programs, as described below.

3.2.3 Mutant pool and FEP matrix

We used the Proteum mutation system [2] to obtain mutant versions of our subject programs; this process produced between several and several dozen mutations of each executable statement in each subject program. We treat the set of mutants for each program as the *mutant pool* for that program; Figure 1 lists the size of these mutant pools. For each program, we used its mutant pool and test pool to evaluate the fault exposure potential of each statement in the program, as described in Section 2. We thereby generated an *FEP matrix* for each program, which records the FEP estimates for each executable statement in that program.

[2]We allowed 17 edges reachable only on `malloc` failures to remain unexercised.

3.2.4 Confidence levels

To address our research questions, we required FEPC-adequate test suites at several confidence levels. Since confidence level is a continuous variable, for our experiment we must sample confidence level. Given the relationship depicted in Figure 1 in Section 2, we judged it sufficient to sample infrequently for low confidence levels, but more frequently for higher confidence levels; this led us to select confidence levels 0.1, 0.4, 0.6, 0.8, 0.9, 0.95, and 0.995.

3.2.5 Test suites

To address our research questions we need to be able to compare the efficacies of FEPC-adequate test suites with the efficacies of some control group of test suites that do not incorporate FEP estimates. However, we must be careful in choosing a candidate for such a comparison, because there are many factors that can affect a test suite's efficacy. To assess the effects of incorporating an estimate of fault exposure probability into the statement-coverage adequacy criterion, we must strictly control for those factors. Thus, we considered three different control groups of test inputs.

Group 1: statement-coverage-adequate test suites. It is natural to consider statement-coverage-adequate test suites as a comparison candidate because both FEPC and statement-coverage adequacy focus on statement execution; moreover, FEPC adequacy yields, by definition, a statement-coverage-adequate test suite. A problem with this approach, however, involves test suite size. Obviously, test suite size can affect fault-detection effectiveness, and if two test suites have different sizes, it is difficult to determine whether differences in the efficacy of those test suites are due to the coverage they obtain, or simply to their differing sizes. To assess whether differences in efficacy are due to the use of FEPC adequacy, we must control for size.

Group 2: random test suites. A second option involves comparing the efficacies of FEPC-adequate test suites to the efficacies of equivalently sized random test suites. This approach controls for the size of the test suites, and thus lets us assess efficacy independent of test suite size. A problem with this approach, however, involves code coverage effects. FEPC-adequate test suites necessarily execute each executable statement at least once; thus, they are statement-coverage-adequate. Randomly selected test suites, of course, may not be statement-coverage-adequate. Thus, if a comparison of FEPC-adequate to randomly selected test suites reveals differences in efficacy, we will not be able to distinguish gains that might be yielded by FEPC adequacy from those that are due simply to achieving statement coverage.

Group 3: augmented statement-coverage-adequate test suites. Since FEPC-adequate test suites are statement-cover-age-adequate, we can construct an FEPC-adequate test suite by first constructing a minimal statement-coverage-adequate test suite T_k,[3] and then greedily selecting test cases from the test pool, adding them to the suites if they cover additional hit number requirements, until the hit number

[3]Use of minimal test suites eliminates another threat to validity: the initial statement-coverage-adequate test suite might itself be an "over-qualified" FEPC-adequate test suite. In that case, our comparison would not let us judge whether FEPC adequacy affects efficacy.

requirements of every executable statement (for the confidence level of interest) are satisfied. We can then construct a second, control test suite, by beginning with T_k, and randomly adding test cases to T_k without thought for coverage until it attains the same size as the FEPC-adequate suite. This approach creates a control group of *augmented statement-coverage (ASC)* test suites that are statement-coverage-adequate, yet of the same sizes as their corresponding FEPC-adequate test suites. The approach thus controls for both the effects of test suite size, and statement coverage adequacy. Using these ASC test suites as a control group in our experiments, together with appropriate statistical comparison techniques, we can be much more certain that differences in efficacy, if found, are attributable to the use of fault exposure probability estimates. Thus, in these experiments, we employ ASC test suites.

In our experiments we refer to an FEPC-adequate test suite T_k and its corresponding ASC suite (the suite created from the same statement-adequate base as T_k) as a *test suite pair*. For each program and each confidence level, we generated 1000 (FEPC-adequate, ASC) *test suite pairs*. Given our eight programs and seven confidence levels, this entailed the generation of $8 \times 7 \times 1000 = 56,000$ test suite pairs.

3.2.6 Fault sets

To measure test suite efficacy we required fault sets. For each program we considered three such sets.

Original fault sets. The Siemens researchers seeded the Siemens programs with faults; these faults were intended to be as "realistic" as possible, based on the researchers' experience with real programs. In contrast, `space` has 38 faults, including 33 faults discovered during its development and 5 discovered subsequently by the authors of this paper. The number of faults in the original fault set for each program is given in Table 1 in the rightmost column.

Mutation fault sets. Although the original fault sets contained a selection of both real and "realistic" faults, the sets of faults are somewhat small. To enlarge our focus, we considered a second fault set constructed from the mutations created by Proteum. We obtained this set by randomly selecting, for each program, 200 mutants from the mutant pool for that program. We restricted our selection to mutants that were known to be non-equivalent: that is, mutants for which there existed at least one test case, in the test pool for the program, that exposed that mutant.

Tough fault sets. Pilot studies suggested that FEPC-adequate test suites might attain greater efficacy when applied to faults that are difficult to detect. Thus, in our experiments, we utilized a third group of *tough fault sets*, consisting of relatively difficult to detect faults. We obtained this set by randomly selecting mutants, from the mutant pool, that had FEP estimates less than 0.2, but greater than 0.0 (and thus not equivalent).[4] We selected tough fault sets of size 200 for each program except `schedule`, for which there were only 90 qualified mutants.

[4]We define the FEP estimate for a *mutant* as the number of times the mutant is exposed by test cases in the test pool, divided by the number of test cases in the test pool that execute the statement containing the mutant.

3.3 Experiment Design

3.3.1 Variables

The experiment manipulated three independent variables:

1. The subject program (8 programs).
2. The confidence level (7 different confidence levels).
3. The fault set (3 different fault sets for each program).

We measured 2 dependent variables:

1. Fault-detection effectiveness (efficacy measure).
2. Test suite size.

3.3.2 Design

The experiment used an $8 \times 7 \times 3$ factorial design with 1000 paired efficacy measures per cell; the three categorical factors were *program*, *confidence level*, and *fault set*. For each program P and confidence level c, we ran our 1000 test suite pairs on each fault set. This yielded 168,000 paired efficacy measures; these formed the data set for our analysis.

3.4 Analysis and Results

The three subsections that follow analyze the data obtained using each of the different fault set types. Section 4 presents further discussion of these results and further observations.

3.4.1 Original Fault Sets

Figure 2 depicts average efficacy values of the paired FEPC-adequate and ASC test suites measured against the *Original Fault Sets* over the seven confidence levels. Each graph depicts results for one subject program. In the graphs, each plotted point represents the mean of the 1000 efficacy values collected at a given confidence level for the FEPC adequate test suites (filled diamond plot symbol) and ASC adequate test suites (hollow circle plot symbol). The graphs depict the differences in fault-detection effectiveness between FEPC-adequate and ASC test suites.

As the graphs show, the average efficacy of FEPC-adequate suites and ASC suites increases as confidence level increases. This increase occurs for all programs, albeit at different rates. For `print_tokens2`, `schedule2`, `tcas`, and `tot_info`, the average efficacy values of the FEPC-adequate suites are noticeably larger than those of the ASC suites as confidence level ranges from 0.4 to 0.995. For `print_tokens` and `schedule`, the average efficacy values of the FEPC-adequate suites are somewhat larger than those of the ASC suites as confidence level ranges from 0.6 to 0.95. For the larger program `space`, the average efficacy values of FEPC-adequate suites are larger than those of the ASC suites at all confidence levels. For `replace`, the average efficacy values of FEPC-adequate suites appear to be either smaller than or equal to those of the ASC suites.

Our hypothesis is that *the fault-detection effectiveness of FEPC-adequate suites will be better than the fault-detection effectiveness of ASC suites.* Consequently we expect to find positive *mean differences* (that is, the difference between the average efficacies of the FEPC-adequate suites and ASC suites) from our data. To formally assess which mean differences are statistically significant, paired t-tests were run. Mean differences where the t-test ρ (rho) value is less than or equal to 0.05 are deemed statistically significant.

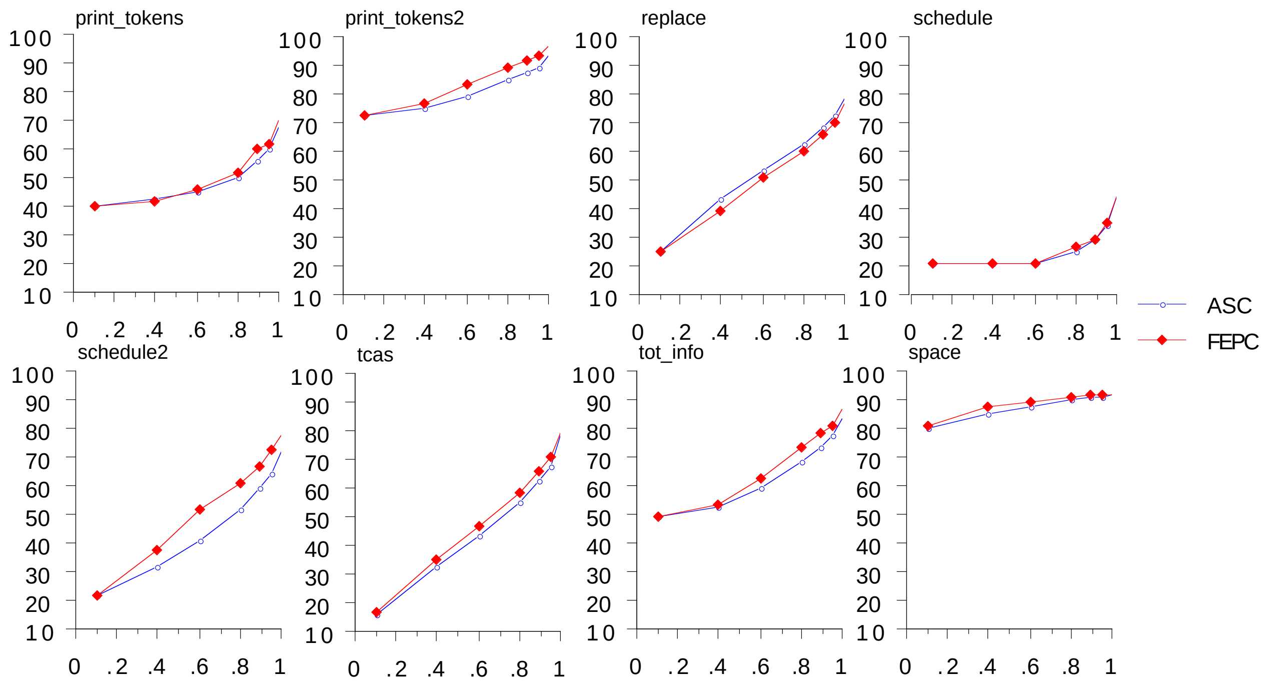

Figure 2: Average efficacy values of FEPC-adequate and ASC test suites, per program, run against the *Original Fault Sets*. Efficacy is shown along the vertical axis and confidence level along the horizontal axis.

program	*cl=0.1* m.d.(%)	*cl=0.4* m.d.(%)	*cl=0.6* m.d.(%)	*cl=0.8* m.d.(%)	*cl=0.9* m.d.(%)	*cl=0.95* m.d.(%)	*cl=0.995* m.d.(%)	*total* m.d.(%)
print_tokens	0.0	-0.1	**0.5**	**1.1**	**3.8**	**2.1**	**2.7**	**1.5**
print_tokens2	*0.04*	**2.0**	**3.9**	**4.8**	**4.2**	**3.8**	**3.6**	**3.2**
replace	-0.5	-3.9	-2.4	-2.6	-2.8	-2.3	-1.6	-2.3
schedule	0.0	0.0	**0.2**	**1.4**	**0.5**	**0.5**	**0.1**	**0.4**
schedule2	**0.2**	**5.9**	**10.3**	**9.2**	**8.0**	**8.4**	**5.2**	**6.7**
tcas	**0.3**	**2.9**	**3.2**	**3.1**	**3.4**	**3.2**	**1.2**	**2.5**
tot_info	0.0	**1.2**	**4.0**	**5.3**	**5.4**	**3.5**	**2.9**	**3.2**
space	**0.7**	**1.9**	**1.6**	**1.3**	**1.0**	**0.8**	**0.4**	**1.1**
total	**0.1**	**1.3**	**2.7**	**2.9**	**2.9**	**2.5**	**1.8**	**2.0**

Table 2: Mean differences between FEPC-adequate and ASC test efficacies for *Original Fault Sets*. Italicized entries are not statistically significant ($\alpha = .05$).

Table 2 displays the mean differences in efficacy values (as percentages) between FEPC-adequate and ASC test suites, by program, with an all programs total. The three classes of table entries are distinguished by different type styles. Boldfaced entries indicate statistically significant results supporting our hypothesis (mean difference > 0, with $\rho \leq 0.05$). Entries in standard type indicate results that are contrary to the hypothesis (mean difference ≤ 0, with $\rho \leq 0.05$). Italicized entries indicate results that are statistically not significant ($\rho > 0.05$) and hence inconclusive.

The bottom-right entry of the table contains a statistically significant positive mean difference derived from analyzing all 56,000 efficacy measure pairs as one data set. The result supports the hypothesis; this suggests that overall, the fault detection effectiveness of FEPC-adequate suites was better than that of their corresponding ASC suites.

The entries in the right-most column of the table contain the mean differences calculated from the 7000 efficacy measure pairs collected (across all seven confidence levels) on that program. The results indicate support for the hypothesis on seven of the eight programs. The results on `replace`, however, are contrary to the hypothesis.

The bottom row of the confidence level columns contain the mean differences calculated from the 8000 efficacy measure pairs collected (across all eight programs) at that level. Overall, each entry indicates supportive results: at each confidence level, FEPC-adequate suites outperform ASC suites. The mean difference values in this row, from left to right, form a single-peak curve, increasing to confidence levels 0.8 and 0.9, and declining thereafter.

The other (interior) entries in the table present the results from the 1,000 paired efficacy measures collected for each program at each specified confidence level. Of these 56 results, 43 entries (77%) are supportive, 12 entries are contrary, and one is not statistically significant. `Replace` exhibits contrary or insignificant results at every confidence level. Five of the eight programs exhibit contrary or insignificant results at confidence level 0.1.

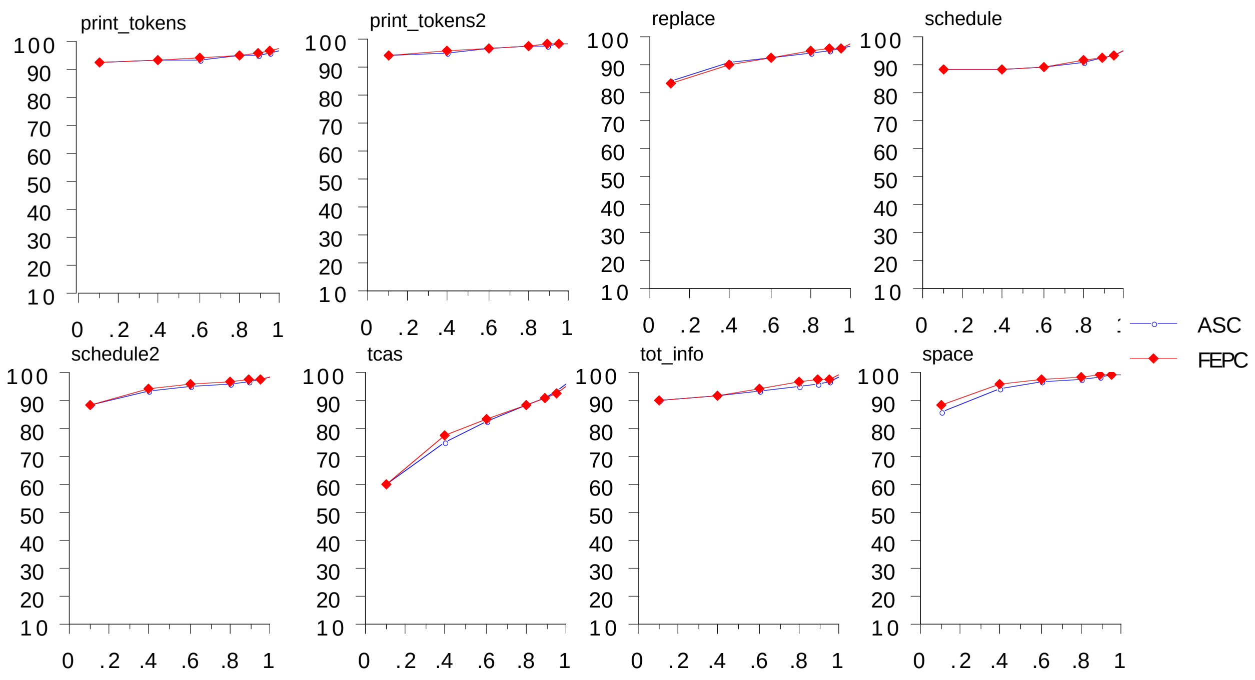

Figure 3: Average efficacy values of FEPC-adequate and ASC test suites, per program, run against the *Mutant Fault Sets*. Efficacy is shown along the vertical axis and confidence level along the horizontal axis.

	cl=0.1	*cl=0.4*	*cl=0.6*	*cl=0.8*	*cl=0.9*	*cl=0.95*	*cl=0.995*	*total*
program	m.d.(%)	m.d.(%)	m.d.(%)	m.d.(%)	m.d.(%)	m.d.(%)	m.d.(%)	m.d.(%)
print_tokens	0.0	*0.03*	**0.3**	**0.2**	**0.5**	**0.5**	**0.4**	**0.3**
print_tokens2	*0.01*	**0.3**	**0.6**	**0.4**	**0.3**	**0.3**	**0.2**	**0.3**
replace	-0.7	-1.1	-0.02	**0.4**	**0.6**	**0.6**	**0.4**	*0.01*
schedule	0.0	0.0	**0.1**	**0.5**	**0.2**	**0.3**	**0.03**	**0.2**
schedule2	**0.3**	**0.6**	**1.0**	**0.8**	**0.8**	**0.7**	**0.4**	**0.7**
tcas	0.0	**0.5**	**0.8**	**1.2**	**1.1**	**0.8**	**0.7**	**0.7**
tot_info	0.0	**0.4**	**0.8**	**0.9**	**1.0**	**1.0**	**0.8**	**0.7**
space	**3.1**	**1.6**	**1.2**	**0.7**	**0.6**	**0.4**	**0.1**	**1.1**
total	**0.3**	**0.3**	**0.6**	**0.6**	**0.6**	**0.6**	**0.4**	**0.5**

Table 3: Mean differences between FEPC-adequate and ASC test efficacies for *Mutation Fault Sets*. Italicized entries are not statistically significant ($\alpha = .05$).

3.4.2 Mutation Fault Sets

Figure 3 depicts average efficacy values of FEPC-adequate and ASC test suites measured against the *Mutation Fault Sets*. Comparing Figures 2 and 3 it appears that the same general trends in efficacy occur as confidence level changes. However, for all programs, the average efficacy values of FEPC-adequate and ASC test suites are larger for the *Mutation Fault Sets* than for the *Original Fault Sets*: particularly in the cases of **print_tokens**, **replace**, **schedule**, **schedule2**, and **tcas**. The difference suggests that faults in the *Mutation Fault Sets* are easier to detect than those in the *Original Fault Sets*.

The mean differences obtained against the *Mutation Fault Sets* are shown in Table 3 (using the same type style conventions used in Table 2.) At the overall level (bottom-right table entry), results continue to support our hypothesis, but at a mean difference lower than that observed against the *Original Fault Sets* (0.5% as opposed to 2.0%).

At the overall program level (right column), results support our hypothesis on seven of the eight programs, with results on **replace** showing no significant differences. On all of the programs, however, the mean differences are closer to 0.

At the overall confidence level (bottom row), all entries continue to indicate supportive results. For all but one entry ($cl = 0.1$), however, the mean differences observed are lower than those observed with the *Original Fault Set*, suggesting less gain in fault-detection as a result of employing FEPC-adequacy with these fault sets.

The individual table entries, for the most part, reflect the same movement toward 0.0 difference typically exhibited at the overall program and overall confidence levels.

3.4.3 Tough Fault Sets

Figure 4 depicts average efficacy values of FEPC-adequate and ASC test suites measured against the *Tough Fault Sets*. The graphs again exhibit efficacy trends across confidence levels similar to those observed on the other fault sets. In general, however, the mean differences in efficacy values are higher than those displayed in Figure 3, presumably reflecting the differences in fault difficulty between these fault sets.

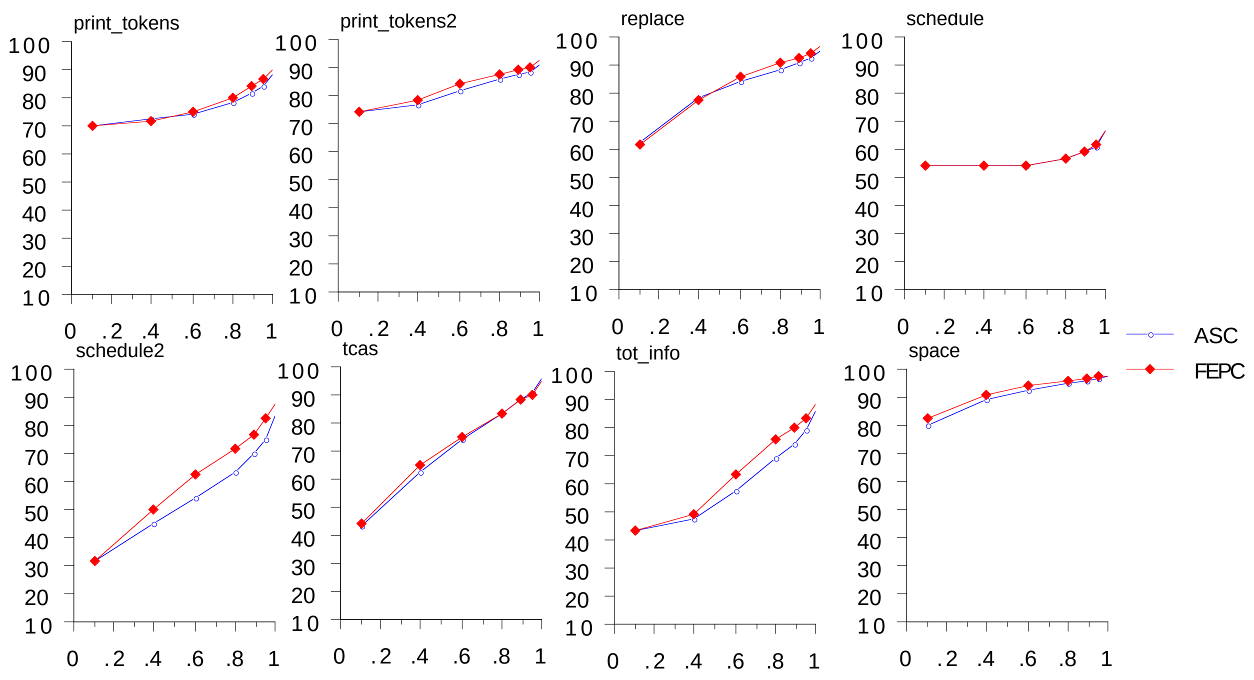

Figure 4: Average efficacy values of FEPC-adequate and ASC test suites, per program, run against the *Tough Fault Sets*. Efficacy is shown along the vertical axis and confidence level along the horizontal axis.

program	*cl=0.1* m.d.(%)	*cl=0.4* m.d.(%)	*cl=0.6* m.d.(%)	*cl=0.8* m.d.(%)	*cl=0.9* m.d.(%)	*cl=0.95* m.d.(%)	*cl=0.995* m.d.(%)	*total* m.d.(%)
print_tokens	0.0	**0.6**	**0.7**	**1.1**	**2.7**	**2.6**	**1.9**	**1.2**
print_tokens2	*0.02*	**1.4**	**2.6**	**2.0**	**1.8**	**1.6**	**1.1**	**1.5**
replace	-1.0	-0.9	**1.3**	**2.3**	**2.1**	**2.0**	**1.4**	**1.0**
schedule	0.0	0.0	**0.1**	**0.1**	*0.003*	**0.2**	*0.02*	**0.1**
schedule2	**0.2**	**5.3**	**8.5**	**8.2**	**7.0**	**7.5**	**4.7**	**5.9**
tcas	**0.4**	**2.4**	**1.2**	-0.3	-0.1	-0.9	-0.8	**0.3**
tot_info	0.0	**1.9**	**5.3**	**6.8**	**5.9**	**4.4**	**2.9**	**3.9**
space	**2.3**	**1.7**	**1.2**	**1.0**	**1.0**	**0.9**	**0.6**	**1.2**
total	**0.2**	**1.4**	**2.6**	**2.6**	**2.5**	**2.3**	**1.5**	**1.9**

Table 4: Mean differences between FEPC-adequate and ASC test efficacies for *Tough Fault Sets*. Italicized entries are not statistically significant ($\alpha = .05$).

The mean differences obtained against the *Tough Fault Sets* are shown in Table 4. The mean difference values at the overall program level indicate that on seven programs, FEPC-adequate suites yielded better efficacies against the *Tough Fault Sets* than against the *Mutation Fault Sets*, while for `tcas` results were slightly worse. The results in the last row of Table 4 show that at the overall confidence level, for all levels except 0.1, the mean differences in efficacies measured against the *Tough Fault Sets* were better than those measured against the *Mutation Fault Sets*.

3.4.4 Test Suite Sizes

As expected, our experiment showed that an increase in confidence level resulted in an increase in the size of corresponding FEPC-adequate test suites. Moreover, the rate of increase becomes larger as confidence level increases.

Figure 5 depicts test suite sizes for all eight programs at all seven confidence levels. The individual boxplots show the distribution of test suite sizes for each confidence level. To produce this view of the data it was necessary to normalize the test suite sizes, which varied widely between programs. For each program P we noted the size n of its largest test suite under any confidence level; we then normalized the size of each test suite T for P by calculating the value T/n. Figure 5 plots the distributions of these normalized values within each confidence level. The data shows quadratic growth trends reminiscent of the curves of Figure 1.

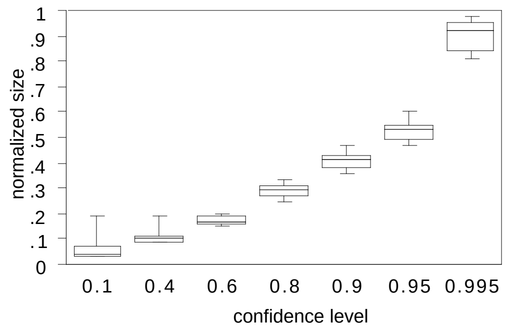

Figure 5: Test suite sizes.

3.5 Threats to Validity

Threats to internal validity are factors that can affect the dependent variables and are out of strict control in the experiment. In this study, we have two major concerns. First, differences among program subjects and in the composition of the test pools may affect results beyond our understanding and ability to control. For example, our test pools are not operational distributions. (2) Second, our method for calculating FEP estimates provides one approach for approximating and using fault exposure probabilities; however, there may be more accurate approaches.

Threats to external validity limit our effectiveness to generalize our results. There are two primary threats to external validity for this study: First, the subject programs are of small and medium size. Complex industrial programs with different characteristics may be subject to different cost-benefit tradeoffs. Second, we used three varieties of fault sets in the experiment; each variety has drawbacks in terms of representativeness. Only the faults for `space` actually occurred in practice, and mutations represent only a relatively small set of the types of possible faults. Also, each fault was considered as the only fault in the program while test cases were running against it. In practice, programs have much more complex error patterns, including faults that interact.

Threats to construct validity arise when measurement instruments do not adequately capture the concepts they are supposed to measure. There are two primary issues to consider. First, efficacy is not the only possible measure of test suite effectiveness. For example, our measures assign no value to subsequent test cases that detect a fault already detected; such inputs may, however, help software engineers isolate the fault, and for that reason might be worth measuring. Second, our efficacy measure does not account for the possibility that faults may have different costs.

4. DISCUSSION

Our results show that the incorporation of fault exposure probability estimates (in the form of FEP estimates) into statement-coverage-adequate test suites can indeed improve the fault detection effectiveness (measured as efficacy) of those test suites. However, these results bear further scrutiny.

First, efficacy results varied widely among the different programs and fault sets; in some cases, results contradicted the suggestion that FEPC-adequate test suites would be more effective than their corresponding ASC suites; in other cases, results showed no significant differences between the suites. On all programs other than `replace`, FEPC-adequate suites were more effective than ASC suites for all three fault sets. On `replace`, in contrast, FEPC-adequate suites were *less* effective overall than ASC suites for two of the three fault sets. `Schedule` is an interesting case: from the efficacy graphs and paired t-test results for `schedule`, we can see that `schedule`'s FEPC-adequate test suites and ASC suites often failed to differ or differed little in efficacy, irrespective of confidence level and fault set.

We believe that there are many factors that may account for these performance differences. One such factor is the range of FEP estimates for the program under test. For example, the test suites for `schedule`, in contrast to those for other programs, are relatively small across confidence levels. Even at confidence level 0.995, the average test suite size for `schedule` is only 17. Checking the hit number requirements for `schedule` under confidence level 0.995, we discovered that most of these were small: only nine of the 281 statements for which hit numbers are calculated possessed hit numbers over five. It seems likely that in such cases, most ASC suites can also satisfy, or nearly satisfy, most hit number requirements, and thus provide efficacy nearly equivalent to that of the corresponding FEPC-adequate suites.

Confidence level also affects results. Under confidence levels 0.1 and 0.4, the efficacies of ASC and FEPC-adequate suites often do not significantly differ, or differ only slightly. However, at these confidence levels, most hit number requirements are small, and minimized statement coverage test suites may themselves be nearly FEPC-adequate. As confidence levels increase, the hit number requirements for many statements increase dramatically, reducing the likelihood that random augmentation of test suites will possess the "extra intelligence" inherent in adding test cases that focus on statements where faults are more likely to hide.

Results also vary with type of faults. On all programs except `schedule` and `tcas`, the efficacy benefits for FEPC-adequate suites are greater for harder-to-detect faults than for easier-to-detect faults. This result supports the theory underlying FEPC-adequacy: probabilistically, hard-to-detect faults are expected to be located in statements that have low FEP estimates and, consequently, high hit numbers; FEPC-adequate suites should be more effective than ASC suites at exercising these statements. Our observations suggest then, that our estimate of fault exposing potential has had been somewhat successful at capturing the underlying probabilities.

We expect that factors such as program, confidence level, and fault type interact in complex fashions. For practical purposes, we would like to better understand these factors and their interaction, so that we could predict whether and when the incorporation of estimates of fault exposing potential would be useful. Future study in this area is necessary.

Perhaps our most interesting observation, however, concerns cost-benefits tradeoffs involving FEPC-adequate test suites. Consider the graph of efficacy results for `schedule2` in Figure 4. In this case, at confidence level 0.6, the mean efficacy of the ASC suites is 52.5% and the mean efficacy of the FEPC-adequate suites is 61%. However, it is clear from the graph that ASC suites for (approximately) confidence level 0.8 achieve the same efficacy as FEPC-adequate suites at level 0.6. This example illustrates a more general observation: for any FEPC-adequate test suite T, there exists some ASC suite (some statement adequate test suite to which n test cases have been randomly added), that achieves the same average efficacy as T.

Since the cost of obtaining FEPC-adequate test suites may be high, the fact that test suites of equivalent efficacy can be generated by random addition of a sufficient number of test cases is significant. In this case, the relative cost-benefits of the two types of test suites depend on both (1) the cost of the analysis necessary to obtain the FEPC-adequate test suites, and (2) the costs of running test cases. If test case execution is sufficiently inexpensive, randomly augmented ASC suites would be more cost-effective; however, if test case execution is sufficiently expensive, incurring the analysis costs necessary to obtain smaller, FEPC-adequate test suites would be more cost-effective.

This observation should be further qualified. Our examinations of test suite size show that as confidence level increases, the size of FEPC-adequate test suites increases dramatically. At higher confidence levels, the number of test cases that must be randomly added to an ASC suite to achieve the efficacy of some FEPC-adequate suite is *much higher* than at lower levels. Therefore, when higher confidence is required, there is greater potential for FEPC-adequate suites to be more cost-effective (depending on the relative costs of test execution and FEP estimation analysis) than ASC suites.

Finally, whether the efficacy gains that may be achieved by FEPC-adequate test suites are worthwhile is also a matter of cost-benefits tradeoffs involving several factors, including (1) the relative costs of analysis and test execution, (2) the costs of failing to detect faults, and (3) the relative gains that could be achieved by employing resources on other validation activities. Consider the results for the most realistic program utilized in our studies, `space`, against the set of real faults for that program. In this case, FEPC-adequate suites detected only 1.1% more faults than their corresponding ASC suites. A 1.1% increase in fault-detection for a word processing system whose test cases are relatively inexpensive to execute, obtained through expensive analysis, would most likely not be cost-effective. A 1.1% increase in fault-detection in software that will operate in a satellite, whose test cases may be relatively expensive to execute, may be cost-effective despite expensive analysis.

5. CONCLUSIONS

Although several researchers have hypothesized that the incorporation of fault exposure probability estimates into test data adequacy criteria could improve the fault-detection effectiveness of test suites, this suggestion has not previously been empirically investigated. This paper has presented the first formal experiment directed at this hypothesis.

The particular technique that we have used to estimate fault exposure probabilities in this experiment is expensive due to its reliance on mutation analysis and would be impractical for most real applications. However the goal of our experiment was not to evaluate a technique, but rather, to answer the important initial question of whether, if an effective technique existed, it could be cost-effective.

Our results suggest that benefits can indeed accrue from the incorporation of fault exposure potential estimates into test adequacy criteria, depending on the relative costs of estimation, test execution, and undetected faults. However, the potential benefits also vary with several other factors including program, required confidence level, and fault type. Further, the overall improvements in fault-detection effectiveness that we observed under our particular approach are not as large as we might wish. We hope that by further study of the factors that influence fault exposure potential, and by considering other sources of estimates of that potential (as discussed in Section 2.2) and improving those estimates, we can further improve the effectiveness of FEPC-adequate test suites and the range of applications in which they are cost-effective. This work provides impetus for that research.

ACKNOWLEDGEMENTS

This work was supported in part by NSF under awards CCR-9703108 and CCR-9707792 to Oregon State University. Siemens Corporate Research shared the Siemens programs. Alberto Pasquini, Phyllis Frankl, and Filip Vokolos shared `space` and its tests.

REFERENCES

[1] L. Clarke, A. Podgurski, D. Richardson, and S. Zeil. A formal evaluation of data flow path selection criteria. *IEEE Trans. on Softw. Eng.*, SE-15(11):1318–1332, Nov. 1989.

[2] M. E. Delamaro and J. C. Maldonado. Proteum–A Tool for the Assessment of Test Adequacy for C Programs. In *Proc. Conf. on Performability in Computing Sys. (PCS 96)*, pages 79–95, July 25–26 1996.

[3] R. A. DeMillo, R. J. Lipton, and F. G. Sayward. Hints on test data selection: Help for the practicing programmer. *Computer*, 11(4):34–41, Apr. 1978.

[4] T. Goradia. Dynamic impact analysis: A cost-effective technique to enforce error-propagation. In *ACM Int'l. Symp. on Softw. Testing and Anal.*, pages 171–181, June 1993.

[5] D. Hamlet and J. Voas. Faults on its sleeve:. In *ACM Int'l. Symp. on Softw. Testing and Anal.*, pages 89–98, June 1993.

[6] R. G. Hamlet. Testing programs with the aid of a compiler. *IEEE Trans. Softw. Eng.*, SE-3(4):279–290, July 1977.

[7] R. G. Hamlet. Probable correctness theory. *Info. Processing Letters*, 25:17–25, Apr. 1987.

[8] W. E. Howden. Weak mutation testing and completeness of test sets. *IEEE Trans. Softw. Eng.*, SE-8:371–379, July 1982.

[9] M. Hutchins, H. Foster, T. Goradia, and T. Ostrand. Experiments on the effectiveness of dataflow- and controlflow-based test adequacy criteria. In *Proc. 16th Int'l. Conf. on Softw. Eng.*, pages 191–200, May 1994.

[10] T. M. Khoshgoftaar and J. C. Munson. Predicting software development errors using complexity metrics. *J. on Selected Areas in Comm.*, 8(2):253–261, Feb. 1990.

[11] L. J. Morell. A theory of fault-based testing. *IEEE Trans. Softw. Eng.*, 16(8):844–57, Aug. 1990.

[12] A. J. Offutt, A. Lee, G. Rothermel, R. Untch, and C. Zapf. An experimental determination of sufficient mutation operators. *ACM Transactions on Software Engineering and Methodology*, 5(2):99–118, Apr. 1996.

[13] T. Ostrand and M. Balcer. The category-partition method for specifying and generating functional tests. *Comm. of the ACM*, 31(6), June 1988.

[14] D. Perry and G. Kaiser. Adequate testing and object-oriented programming. *J. O.-O. Prog.*, 2, Jan. 1990.

[15] D. Richardson and M. Thompson. An analysis of test data selection criteria using the RELAY model of fault detection. *IEEE Trans. on Softw. Eng.*, pages 533–553, June 1993.

[16] G. Rothermel, R. H. Untch, C. Chu, and M. J. Harrold. Test case prioritization: An empirical study. In *Proc. Int'l. Conf. Softw. Maint.*, Aug. 1999.

[17] M. Thompson, D. Richardson, and L. Clarke. An information flow model of fault detection. In *ACM Int'l. Symp. Softw. Testing and Anal.*, pages 182–192, June 1993.

[18] J. Voas. PIE: A dynamic failure-based technique. *IEEE Trans. on Softw. Eng.*, pages 717–727, Aug. 1992.

[19] F. I. Vokolos and P. G. Frankl. Empirical evaluation of the textual differencing regression testing technique. In *Proc. Int'l. Conf. Softw. Maint.*, pages 44–53, Nov. 1998.