AN ABSTRACT OF THE THESIS OF

Wei Chen for the degree of Master of Science in Computer Science presented on June 30, 2000.

Title:

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

Abstract approved: Redacted for Privacy

Gregg Rothermel

Researchers have hypothesized that if we could estimate the probability that a fault in a code component will a cause a failure, we could use this estimate to improve the fault-detection effectiveness of code-coverage-based testing. If this hypothesis could be supported, it would motivate further research in this area and could lead to techniques that would help testers distribute testing resources more effectively and improve the quality of testing. In this research, we developed a new test adequacy criterion, which incorporates fault exposure potential estimates into statement-coverage requirements. We conducted empirical studies to investigate the fault-detection effectiveness of this new criterion. The results of our studies show that the incorporation of fault exposure potential information into the statement-coverage test adequacy criterion can indeed improve its fault-detection effectiveness. However, the effects of incorporating the estimates vary with the program under test, the nature of faults contained in the program, and the level of confidence required of the testing. The overall

improvements in fault-detection effectiveness that we observed under our initial approach were not as large as we might wish, but by improving the method for estimating fault exposure potential we can obtain better results. The results of this research provide impetus for future studies in this area.

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

by

Wei Chen

A Thesis

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

Completed June 30, 2000
Commencement June 2001

Master of Science thesis of Wei Chen presented on June 30, 2000

APPROVED:

Redacted for Privacy

Major Professor, representing Computer Science

Redacted for Privacy

Chair of the Department of Computer Science

Redacted for Privacy

Dean of the Graduate School

I understand that my thesis will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my thesis to any reader upon request.

Redacted for Privacy

Wei Chen, Author

ACKNOWLEDGMENTS

I would like to give my special thanks to my major professor, Dr. Gregg Rothermel, who provided much encouragement, guidance, and advice on my research work during the past year, and provided great help with the preparation this paper.

Thanks to Dr. Michael Quinn, for serving as my minor professor, and to Dr. Curtis Cook and Dr. David Birkes, for serving on my committee.

Thanks also to Dr. Roland Untch of Middle Tennessee State University, who provided the mutation data necessary for my research work. He also assisted with the statistical analysis on the experimental results in this research work.

Jefferey von Ronne and Chengyun Chu provided some important software tools needed in my research work.

Jim Law answered my many questions about the Aristotle system.

Thanks to everyone.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

LIST OF TABLES

EMPIRICAL STUDIES OF THE EFFECTS OF INCORPORATING FAULT EXPOSURE POTENTIAL ESTIMATES INTO A TEST DATA ADEQUACY CRITERION

Chapter 1

INTRODUCTION

The goal of software testing is to show the presence of possible faults in the software. Since computer technology is now playing a key role in everyday life, a software failure can cause great damage, or even the loss of human life. As a method for improving the quality of software, software testing's importance is increasing dramatically.

It is generally accepted that software testers need some test data adequacy criterion with which to evaluate whether a set of test data is sufficient and to guide 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. [2, 12, 17]) and are often used in the software industry.

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 [6, 7, 14, 18, 21, 22].

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 [6, 7, 8, 20, 22]. 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 [7, 8, 22].

This suggestion is intriguing; however, in our search of the research literature, we could discover no empirical studies that have directly assessed it. Voas [22] reports results of a study assessing the correlation between PIE (propagation, infection, and execution) analysis sensitivity estimates and failures observed in random testing. Goradia [6] 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 can 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. Therefore, in this work we conducted a controlled experiment to investigate the effects of incorporating fault

exposure potential estimates into a test 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 also highlight several interesting cost-benefits tradeoffs with respect to the incorporation of the estimate.

Further analysis of the results of this experiment indicates that there may be several ways to improve this approach. We conducted another experiment to investigate one of these potential improvements, and the results are positive.

The structure of this thesis is as follows. In the next chapter we discuss testing methods, code-coverage-based test criteria, fault exposure, and related work. Chapter 3 introduces our method for estimating fault exposure potential, and a test adequacy criterion that incorporates such estimates. Chapter 4 describes our primary empirical study. Our experiment on improving the effectiveness of this approach is presented in Chapter 5. Chapter 6 presents conclusions and discusses possible future work.

Chapter 2

BACKGROUND

2.1 Software Testing

2.1.1 Basics of Software Testing

During a testing process the tested software (or part of it) is executed with some input data, and the output is compared with the hypothetical correct result based on the specifications for the corresponding input data. This procedure is repeated until the testers have enough confidence in the quality of this software. More formally, this process can be described as follows: Let P be a test subject, let D be the input domain of P, and let F be the hypothetical correct version of P (F is called an oracle and may be obtained from the specifications). A test case t is an element of D. A test suite T is a finite subset of D. If for a test case t, $P(t) \neq F(t)$, we say that a failure is demonstrated by t.

In common testing terminology, a failure generally indicates the presence of one or more faults in the software under test. A fault is a mistake that exists in the source code.

Theoretically, an ideal test suite T exists that can detect all the faults that exist in any program. However, in general we can not determine whether a test suite is sufficent to reveal all the faults in a program, or construct such a sufficient suite [10].

Since determining correctness by testing is not feasible in general, testers have focused on building confidence through the use of test adequacy criteria. A test adequacy criterion specifies a condition (or more than one condition) that must be satisfied by a test suite [5]. The purpose of a test adequacy criterion is to guide the construction of a test suite so that it can be sufficient to achieve a certain level of quality in software. Testers may choose and apply one or more test criterion in a real software testing task depending on factors such as budget, human efforts, time, features of the software, and quality requirements.

2.1.2 Software Testing Methods

Normally, software testing methods can be classified as black box or white box approaches. Beside these two classic approaches, there is also a third approach called fault-based testing (the material presented in this section is drawn from [24] and [14]).

2.1.2.1 Black Box Testing

Black box testing methods design test cases without any knowledge of the software's internal structure. The test cases can be derived from specifications, or even be generated randomly. Black box testing is also sometimes called functional testing.

2.1.2.2 White Box Testing

In contrast to black box testing, white box testing designs test cases based on knowledge of the software's internal structure, so it is also called structural testing. Several sophisticated white box testing techniques have been developed. Among them, techniques based on code coverage criteria are the most popular

and successful. These techniques require each valid code component in a program to be executed at least once. The code components can be statements, branches, paths, data dependencies or some similar identifiable entity.

Both black box testing and white box testing have advantages and disadvantages: black box testing is relatively easy to implement, but not as capable of detecting certain classes of faults in the source code as white box testing. White box testing can be very sophisticated, but may not be able to detect errors such as the absence of some required feature or some code component. Thus, in practice, these two approaches are often used together.

2.1.2.3 Fault-based Testing

Unlike black box testing or white box testing, the major objective of fault-based testing is to evaluate the sufficiency of a test suite relative to a specific set of faults [14]. The primary example of this approach is mutation testing [1].

An important assumption for mutation testing is the *coupling assumption*, which states that a test suite that detects small faults in a program is also likely to detect complex, real faults that may occur in that program. In mutation testing, for a program P, a set of small changes in individual statements of P is made and the set of alternative programs obtained are called *mutants*. For a mutant, if any test case in a test suite T can cause it to produce different output from the original program P, we say this mutant has been *killed*, otherwise we say it *survives*. Mutation analysis uses this set of mutants as the set of small faults of P. So, if the test suite T kills most of the mutants of P, then based on the coupling assumption, T should be able to detect most of the faults that may occur in P; if this test suite lets many mutants survive, then it is insufficient and new test cases may need to be added to it.

The assumptions made by mutation testing are controversial, and the process is difficult to manipulate. Because of this, Howden [11] developed a variation of mutation testing called *weak mutation testing* (therefore, testing based on mutation analysis is sometimes called *strong mutation testing*). The major difference between this approach and mutation testing is that in mutation testing, whether a test suite can kill a mutant is judged by whether this test suite can cause the mutated program to produce a different output than the original program, whereas in weak mutation testing a mutant is judged killed if the test suite can cause this mutant to produce a different data state after executing the statement in which the change is located. Weak mutation testing is less expensive than strong mutation testing, but its fault detection capability is not as good as that of strong mutation testing. This is because a test suite that can cause a mutant to produce a different data state may not be able to also cause the mutated program to produce a different final output. Thus, weak mutation can kill mutants more easily than strong mutation and results in weaker test sets.

2.2 Code-Coverage-Based Test Adequacy Criteria and Fault Exposure

An implied assumption of code-coverage-based test adequacy criteria is that all components are equal, so that one execution of each component is enough to guarantee a certain level of the software's quality. However, this assumption is not realistic. It is clear from the research literature that some faults are more easily exposed than others and that some source code components are more easily tested than others [8, 22]. Thus, several researchers (see e.g. [6, 7, 8, 14, 18, 21, 22]) 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]

So, if we know a code component is relatively more difficult to test than other components, we may need to design more than one test case to exercise this component. This means that, if we want to gain a certain level of confidence in the correctness of different code components within the software, we may need to exercise those components different numbers of times. This idea suggests a new type of code-coverage-based test adequacy criteria. This is the main topic of this work.

2.3 Related Work

Voas [22] provides a method called *PIE* (propagation, infection, and execution) analysis, which 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

---

[1] A related issue involves the probability that a component contains a fault (e.g. [13]). We do not address that issue.

obtain these estimates: (1) simple code instrumentation to estimate execution probability; (2) a variant of *weak mutation* [11] in which syntactic changes are applied to x and then the state after x executes is examined 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 presents a systematic method for calculating the execution probability (PE), infection probability (PI), and propagation probability (PP) of a statement s. Once PE, PI, and PP have been estimated, θ, the sensitivity of s, is calculated according to the following formula:

$$\theta = PE \cdot \sigma\,(PI,\ PP)$$

where $\sigma\,(a,\ b) = a - (1 - b)$ if $a - (1 - b) > 0$; otherwise, $\sigma\,(a,\ b) = 0$.

Voas uses an empirical study to show that there can be a significant correlation coefficient between the estimate of the probability of failure measured by random software testing and the probability of failure predicted by the estimates of propagation analysis and execution analysis.

Voas also suggests (following an earlier suggestion by Hamlet [8]) that sensitivity estimates could be used to calculate the number of executions of a component that are required to obtain a certain confidence in that component's correctness. However, neither Voas nor Hamlet further investigate this suggestion.

Chapter 3

PRELIMINARIES

3.1 Fault Exposing Potential Estimation

Voas' PIE analysis has provided a method that could be used to estimate the probability that a fault in a code component will cause a failure. However, for incorporating such information into code-coverage-based test adequacy criteria, this method 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 ap-

proach can overpredict the probability that an arbitrary input will expose a fault and result in low estimates of that probability.

Consider the following example. Suppose for a code component y we obtain 5 estimates of its infection probability (PI) and 5 estimates of its propogation probability (PP), as shown in Table 3.1.

	e1	e2	e3	e4	e5
Pi	1	1	0.5	1	1
PP	1	1	1	1	0.5

TABLE 3.1: PI and PP estimates for code component y.

Using Voas' method, the combined estimate for PI and PP should be:

$$\sigma(PI, PP) = \sigma(0.5, 0.5) = 0.5 - (1 - 0.5) = 0.0$$

Obviously, this estimate is conservative.

Thus, in this work, we adopt a different estimate of the probability that a fault in code component x will cause a failure. For a code component x, we use mutation analysis [4, 9] 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 kill a mutant of x. We call the resulting value the

	m1	m2	m3	m4	m5
t1	1	1	0	1	0
t2	1	1	1	0	0
t3	X	X	X	X	X
t4	1	0	1	0	0
t5	1	0	0	0	0
t6	1	1	0	0	0

TABLE 3.2: Example of FEP estimation for a code component.

Fault-Exposing-Potential (FEP) estimate for x, described more formally by the following equation:

$$FEP_x = \frac{\sum_{i=1}^{k} n_i}{m \times k} \tag{3.1}$$

The following example illustrates. Suppose for a code component x we create 5 mutants, and there are 6 test cases in the test universe. Suppose the result of running these 6 test cases against the 5 mutants is as shown in Table 3.2. In this table, an entry of "1" indicates that the corresponding mutant can be killed by the corresponding test case, an entry of "0" means that the corresponding mutant survives under the corresponding test case, and an entry of "X" means the corresponding test case does not execute the code component containing the mutant. Since test case $t3$ does not execute the component, there are a total of $5 \times 5 = 25$ valid entries. Among these 25 entries, 11 are "1", so the FEP estimate for the code component is $11/25 = 0.44$.

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, there is no such an algo-

rithm to distinguish these mutants in general, also it is not practical to perform this task manually. Thus, a second approach is to treat 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 take this second approach.

We used this FEP estimation method in the empirical study reported in the next chapter. But be aware that our FEP estimation technique is just one approach, and other approaches may be applicable. In chapter 5 we consider one such alternative approach.

3.2 FEPC-Adequacy Test Criterion

3.2.1 Hit Number Calculation

In a code-coverage-based testing task for a program, if we know the FEP estimate for each code component, we can transform this information into the number of test cases that are needed to exercise each code component. For example, if a code component has a very low FEP estimate, this indicates that if a fault exists in this code component, it may be very difficult to detect. In this case we may need to exercise this code component by more test cases than those code components with higher FEP estimates. Also, this interpretation is related to the confidence requirement for the correctness of that program: the higher the confidence requirement, the larger the number of times that a code component should be exercised.

More precisely, following suggestions by Hamlet [8] and Voas [22], we can use an FEP 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 FEP_x 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 FEP_x. 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:

$$hn = \frac{ln(1-c)}{ln(1-FEP_x)} \tag{3.2}$$

For practical purposes two special cases involving equation (3.2) should be considered. First, FEP_x 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 FEP_x is an estimate) is 1. Second, for values of FEP_x 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. We call this final number the *hit number* requirement for x.

To provide a sense of the coverage requirements imposed by such a test adequacy criterion, Figure 3.1 depicts the relationship among FEP estimates, confidence levels, and hit numbers. The figure shows, for four FEP 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 FEP 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 an FEP estimate is low, the hit number required to achieve high confidence is much larger than when an FEP estimate is high.

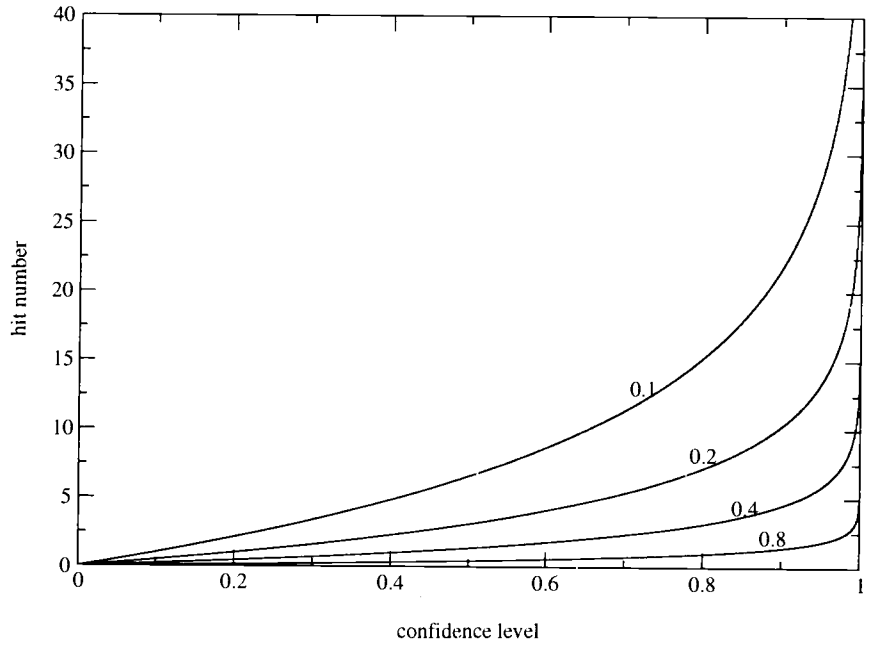

FIGURE 3.1: Hit number versus confidence level for four fault exposure probabilities.

3.2.2 FEPC-Adequacy Test Criteria

Since 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 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. We call this criterion FEPC-adequacy (*Fault-Exposing-Potential-Coverage-adequacy*). 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 type of 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.

Chapter 4

EMPIRICAL STUDY

4.1 Research Questions and Experiment Design Considerations

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

RQ1: Can the incorporation of FEP estimates into statement-coverage test 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: Is program a factor that affect the fault-detection effectiveness of FEPC-adequate test suites?

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

To address our research questions we need to be able to compare the fault-detection effectiveness of FEPC-adequate test suites with that 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 effectiveness. 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.

One major factor that affects the fault-detection effectiveness of a test suite is the size of the test suite. In order to make comparisons of test suites independent of this factor, we must be able to control the size of the test suites compared.

Another factor that we can not ignore involves the effects of statement-coverage adequacy. In the FEPC adequacy test criterion, the hit number requirement for each executable statement is at least one. Therefore, each FEPC-adequate test suite is statement-coverage-adequate. So, the fault-detection effectiveness of these suites may be due to two kinds of causes: their statement-coverage adequacy or their incorporation of FEP estimates. In this study we are interested only in the latter cause, and statement-coverage adequacy is a factor that should be under strict control.

Obviously we can not directly compare FEPC-adequate test suites with statement-coverage-adequate test suites, because by doing so we control the factor of statement-coverage adequacy, but the factor of size is out of control. To control for the size factor we considered using random test suites. A random test suite is composed of randomly selected test cases, and we can easily control its size. However, if we compare FEPC-adequate test suites with equivalently sized random test suites, the factor of statement-coverage adequacy is out of control, simply because we can not guarantee that a random test suite is statement-coverage adequate. So, in this study we used a type of test suite that combines statement-coverage adequacy and randomly selection, so that both factors are under control. Details about the test suites' generation are presented in Section 4.3.5.

The use of confidence levels is a factor unique to FEPC-adequate test suites. We can see from the hit number calculation formula that for an executable statement, under different confidence levels, the hit number requirements may differ. This may result in different FEPC-adequacy test suites under different confidence levels. So we believe that confidence level will have a great impact on FEPC-adequate test suites' fault-detection effectiveness and size. Thus, in this study we performed experiments under several different confidence levels, so that we could investigate the effect of confidence.

4.2 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 $\frac{|F_r|}{|F|} \times 100\%$.

Program	LOCs	Mutant Pool Size	Test Pool Size	Fault Pool Size
print_tokens	712	4057	4130	7
print_tokens2	687	4145	4115	10
replace	563	9622	5542	32
schedule	412	2153	2650	9
schedule2	387	2947	2710	10
tcas	173	2876	1608	41
tot_info	406	5898	1052	23
space	9126	132163	13585	38

TABLE 4.1: Experiment subjects.

4.3 Experiment Instrumentation

4.3.1 Programs

We used eight C programs as subjects (see Table 4.1). 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 [12]; 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.

4.3.2 Test Pool and Test History

For each of the seven Siemens programs the Siemens researchers created a *test*

pool of black-box test cases using the *category partition method* and the Siemens Test Specification Language tool [16]. 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 4.1.

Space has a test pool of 13,585 test cases. The first 10,000 test cases were randomly generated by Vokolos and Frankl [23], the remaining test cases were added by authors of [20], so that most executable branches in the program[1] were exercised by at least 30 test cases.

For our experiment, we considered each program P with test pool U, and recorded, for each test case in U and each statement in P, whether or not that statement was exercised by that test case. We call this information the *test history* of U on P and it was used to create individual test suites for the programs, as described in Section 4.3.5.

4.3.3 Mutant Pool and FEP Matrix

We used the Proteum mutation system [3] 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 4.1 lists the sizes 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

---

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

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.

4.3.4 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 3.1 in Section 3.2, 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.

4.3.5 Test Suites

For this study, we constructed each FEPC-adequate test suite by first constructing a statement-coverage-adequate test suite T_{stmt}, and then using a minimization tool to minimize it to $T_{minstmt}$. Next, we greedily selected test cases from the test pool, adding them to the suites if they covered additional hit number requirements, until the hit number requirements of every executable statement (for the confidence level of interest) were satisfied. This produced an FEPC-adequate test suite. Finally, for each such test suite we constructed an equivalently sized *Augmented-Statement-Coverage* (ASC) test suite, by beginning with $T_{minstmt}$, and randomly adding test cases to $T_{minstmt}$ until it attains the same size as the FEPC-adequate suite.

The method we used to generate our initial statement-coverage-adequate test suites is as follows. We began with a new empty suite, randomly selecting a

test case from the test pool, and if this test case added some statement coverage to the current suite, we added it to the suite. We did this repeatedly until statement coverage had been achieved for the whole program. Obviously, the statement coverage test suite generated by this method may contain redundant test cases, that is, some statements may be exercised more than once by this test suite. Thus, it is possible that the statement-coverage-adequate test suite we generated may be an "over-qualified" test suite, and this may act as an uncontroled factor and cloud our results. We thus minimize this test suite, using the algorithm of [19], to achieve a minimal statement-coverage-adequate test suite.

This approach creates FEPC-adequate test suites, and a control group of 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. In our experiments we refer to an FEPC-adequate test suite T_{FEPC} and its corresponding ASC suite T_{ASC} (the suite created from the same statement-adequate base as $T_{minstmt}$) as a *test suite pair.* The test suite generation procedures are given in detail in Figure 4.1.

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.

```
1.  Algorithm: TestSuitePairGeneration
2.  Input: confidence level C, test history H, test pool U, FEP matrix M
3.  Output: Test suite pair (T_FEPC, T_ASC )
4.  begin
5.    Generate a statement-coverage-adequate test suite T_stmt
6.    Minimize T_stmt to be T_minstmt
7.    T_FEPC = GenFEPCSuite(T_minstmt, C, H, U, M)
8.    T_ASC = GenASCSuite(T_minstmt, T_FEPC, H, U)
9.  end

1.  Procedure: GenFEPCSuite (T_minstmt, C, H, U, M) Output: T_FEPC
2.  begin
3.    calculate hit number requirements for each valid statement
4.    T_FEPC = T_minstmt
5.    while T_FEPC can not satisfy all hit number requirements do
6.      randomly select a test case t from U
7.      if t had not been selected before and t can
        satisfy some hit number requirements not yet satisfied
8.        add t to T_FEPC
9.      endif
10.   endwhile
11. end

1.  Procedure: GenASCSuite (T_minstmt, T_FEPC, C, H, U, M) Output: T_ASC
2.  begin
3.    T_ASC = T_minstmt
4.    while |T_ASC| ≠ |T_FEPC| do
5.      randomly select a test case t from test pool
6.      if t had not been selected before
7.        add t to T_ASC
8.      endif
9.    endwhile
10. end
```

FIGURE 4.1: Algorithm for test suite pair generation.

4.3.6 Fault Sets

In this study we used three kinds of fault sets, as follows.

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 [20].

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 (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), but greater than 0.0 (and thus are not equivalent mutants). We selected tough fault sets of size 200 for each program except `schedule`, for which there were only 90 qualified mutants.

4.4 Experiment Design

4.4.1 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.

4.4.2 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.

4.5 Results and Analysis

The three subsections that follow (4.5.1, 4.5.2, 4.5.3) analyze the data obtained using each of the different fault set types focusing on efficacy. Section 4.5.4 then considers results relevant to test suite size. Following presentation of data we discuss threats to validity for our studies in Section 4.6. Section 4.7 then presents further discussion of these results and further observations.

4.5.1 Original Fault Sets

Figure 4.2 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. 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, although at different rates. For `print_tokens`, `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 `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 their correponding ASC suites.* Consequently we expect to find positive *mean differences* (that is, the difference between the average efficacies of the FEPC-adequate suites and their corresponding ASC suites) from our data. To formally assess which mean

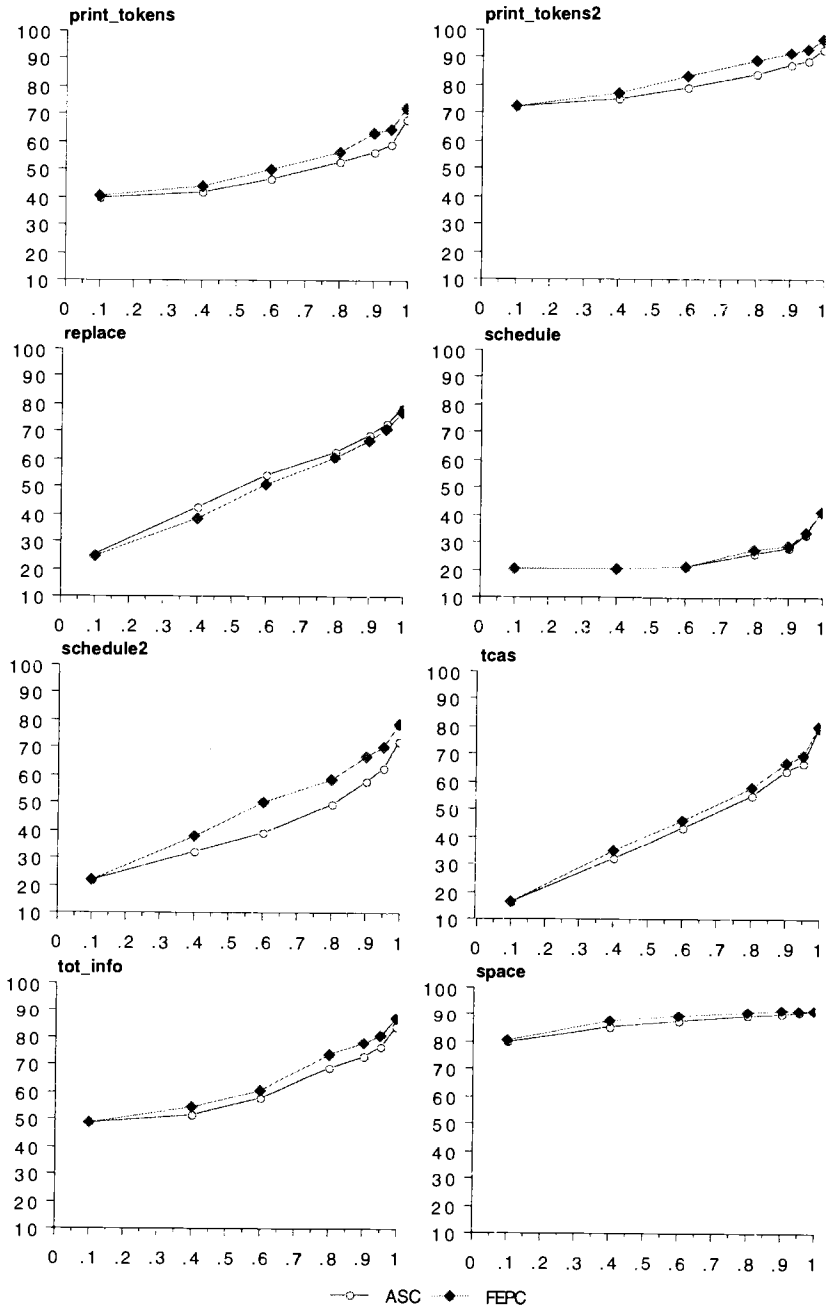

FIGURE 4.2: Average efficacy values of FEPC-adequate and ASC test suites, per program, run against the *Original Fault Sets*. Efficacy is shown along the vertical axis and confidence level along the horizontal axis.

differences are statistically significant, paired t-tests were run. Mean differences where the t-test ρ value is less than or equal to 0.05 are deemed statistically significant.

Table 4.2 displays the mean differences in efficacy values (as percentages) between FEPC-adequate and ASC test suites, and corresponding ρ values, by program, with an all programs total. The three classes of table entries are distinguished by different type styles. Bold-faced 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 our 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 lower-half table (the column labelled total) contain the mean differences calculated from the 7000 efficacy measure pairs collected (across all seven confidence levels) on each 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 (the row entitled "total") of the confidence level columns contains 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

Program	*cl=0.1*		*cl=0.4*		*cl=0.6*		*cl=0.8*	
	mean dif.	ρ value	mean dif.	ρ value	mean dif.	ρ value	mean dif.	ρ value
print_tokens	**0.70**	**<.0001**	**1.70**	**<.0001**	**3.70**	**<.0001**	**3.80**	**<.0001**
print_tokens2	*0.03*	*.1798*	**2.30**	**<.0001**	**3.90**	**<.0001**	**4.60**	**<.0001**
replace	-0.50	<.0001	-3.60	<.0001	-3.30	<.0001	-2.50	<.0001
schedule	0.00	•	0.00	•	**0.20**	**<.0001**	**1.20**	**<.0001**
schedule2	**0.10**	**.0104**	**5.40**	**<.0001**	**11.20**	**<.0001**	**9.10**	**<.0001**
tcas	**0.30**	**.0003**	**2.60**	**<.0001**	**2.40**	**<.0001**	**3.00**	**<.0001**
tot_info	0.00	•	**2.40**	**<.0001**	**3.00**	**<.0001**	**4.70**	**<.0001**
space	**0.50**	**<.0001**	**1.80**	**<.0001**	**1.70**	**<.0001**	**1.40**	**<.0001**
total	**0.10**	**<.0001**	**1.60**	**<.0001**	**2.80**	**<.0001**	**3.20**	**<.0001**

Program	*cl=0.9*		*cl=0.95*		*cl=0.995*		*total*	
	mean dif.	p value	mean dif.	p value	mean dif.	p value	mean dif.	p value
print_tokens	**6.80**	**<.0001**	**4.90**	**<.0001**	**4.10**	**<.0001**	**3.70**	**<.0001**
print_tokens2	**4.40**	**<.0001**	**4.40**	**<.0001**	**3.40**	**<.0001**	**3.30**	**<.0001**
replace	-2.50	<.0001	-2.40	<.0001	-1.60	<.0001	-2.30	<.0001
schedule	**0.60**	**<.0001**	**0.40**	**.0025**	*0.10*	*.2045*	**0.40**	**<.0001**
schedule2	**9.10**	**<.0001**	**7.80**	**<.0001**	**6.20**	**<.0001**	**7.00**	**<.0001**
tcas	**2.60**	**<.0001**	**3.00**	**<.0001**	**0.80**	**<.0001**	**2.10**	**<.0001**
tot_info	**4.30**	**<.0001**	**4.10**	**<.0001**	**2.70**	**<.0001**	**3.10**	**<.0001**
space	**0.90**	**<.0001**	**0.70**	**<.0001**	**0.40**	**<.0001**	**1.00**	**<.0001**
total	**3.30**	**<.0001**	**2.80**	**<.0001**	**2.00**	**<.0001**	**2.30**	**<.0001**

TABLE 4.2: Results of paired t-test on data against *Original Fault Sets.*

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 (a mean difference value and the corresponding ρ value are treated as one result, or say, one entry), 44 entries (79%) are supportive, 10 entries are contrary, and 2 are 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.

4.5.2 Mutation Fault Sets

Figure 4.3 depicts average efficacy values of FEPC-adequate and ASC test suites measured against the *Mutation Fault Sets*. Comparing Figures 4.2 and 4.3 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*.

At the overall program level (right column of lower-half table), 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

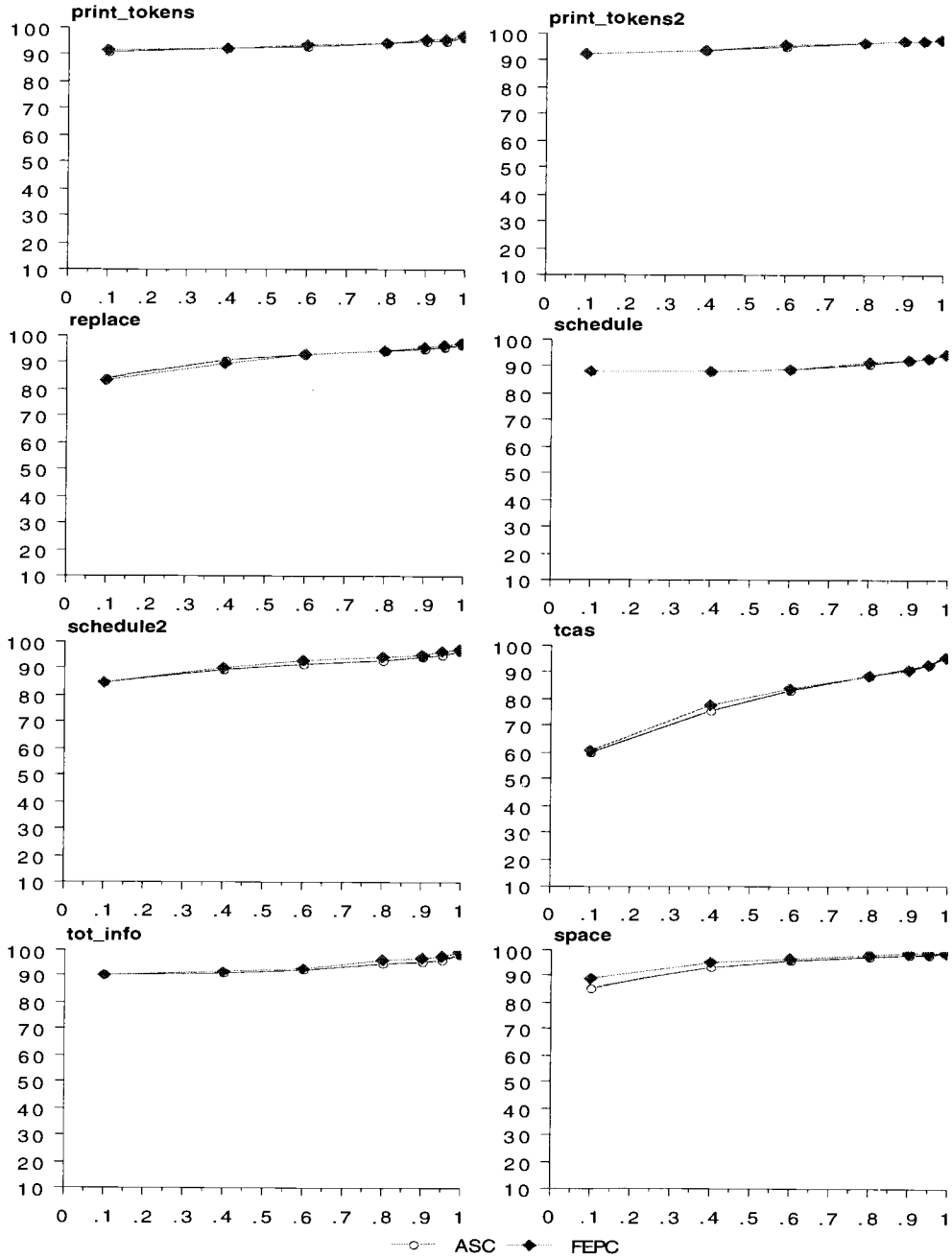

FIGURE 4.3: Average efficacy values of FEPC-adequate and ASC test suites, per program, run against the *Mutation Fault Sets*. Efficacy is shown along the vertical axis and confidence level along the horizontal axis.

Program	*cl=0.1*		*cl=0.4*		*cl=0.6*		*cl=0.8*	
	mean dif.	ρ value	mean dif.	ρ value	mean dif.	ρ value	mean dif.	ρ value
print_tokens	**0.20**	**<.0001**	**0.10**	**<.0001**	**0.40**	**<.0001**	**0.40**	**<.0001**
print_tokens2	*0.04*	*.2484*	**0.40**	**<.0001**	**0.60**	**<.0001**	**0.30**	**<.0001**
replace	-0.70	<.0001	-0.90	<.0001	-.010	.1181	**0.40**	**<.0001**
schedule	0.00	•	0.00	•	**0.20**	**<.0001**	**0.70**	**<.0001**
schedule2	**0.20**	**<.0001**	**0.80**	**<.0001**	**1.40**	**<.0001**	**1.20**	**<.0001**
tcas	**0.30**	**<.0001**	**2.10**	**<.0001**	**0.40**	**<.0001**	**0.50**	**<.0001**
tot_info	0.00	•	**0.60**	**<.0001**	**0.80**	**<.0001**	**1.10**	**<.0001**
space	**3.10**	**<.0001**	**1.70**	**<.0001**	**1.10**	**<.0001**	**0.70**	**<.0001**
total	**0.40**	**<.0001**	**0.60**	**<.0001**	**0.60**	**<.0001**	**0.70**	**<.0001**

Program	*cl=0.9*		*cl=0.95*		*cl=0.995*		*total*	
	mean dif.	p value	mean dif.	p value	mean dif.	p value	mean dif.	p value
print_tokens	**0.70**	**<.0001**	**0.60**	**<.0001**	**0.60**	**<.0001**	**0.40**	**<.0001**
print_tokens2	**0.30**	**<.0001**	**0.20**	**<.0001**	**0.04**	**.0440**	**0.30**	**<.0001**
replace	**0.60**	**<.0001**	**0.50**	**<.0001**	**0.30**	**<.0001**	*0.09*	*.6010*
schedule	**0.30**	**<.0001**	**0.10**	**<.0001**	**0.03**	**.0013**	**0.20**	**<.0001**
schedule2	**1.10**	**<.0001**	**0.90**	**<.0001**	**0.80**	**<.0001**	**0.90**	**<.0001**
tcas	-0.60	<.0001	-0.30	<.0001	-0.60	<.0001	**0.20**	**<.0001**
tot_info	**1.00**	**<.0001**	**0.90**	**<.0001**	**0.80**	**<.0001**	**0.70**	**<.0001**
space	**0.50**	**<.0001**	**0.40**	**<.0001**	**0.10**	**<.0001**	**1.10**	**<.0001**
total	**0.50**	**<.0001**	**0.40**	**<.0001**	**0.30**	**<.0001**	**0.50**	**<.0001**

TABLE 4.3: Results of paired t-test on data against *Mutation Fault Sets.*

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.

4.5.3 Tough Fault Sets

Figure 4.4 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 4.3, presumably reflecting the differences in fault difficulty between these fault sets.

The mean differences obtained against the *Tough Fault Sets* are shown in Table 4.4. 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 4.4 show that at the overall confidence level, for all levels, the mean differences in efficacies measured against the *Tough Fault Sets* were better than those measured against the *Mutation Fault Sets*.

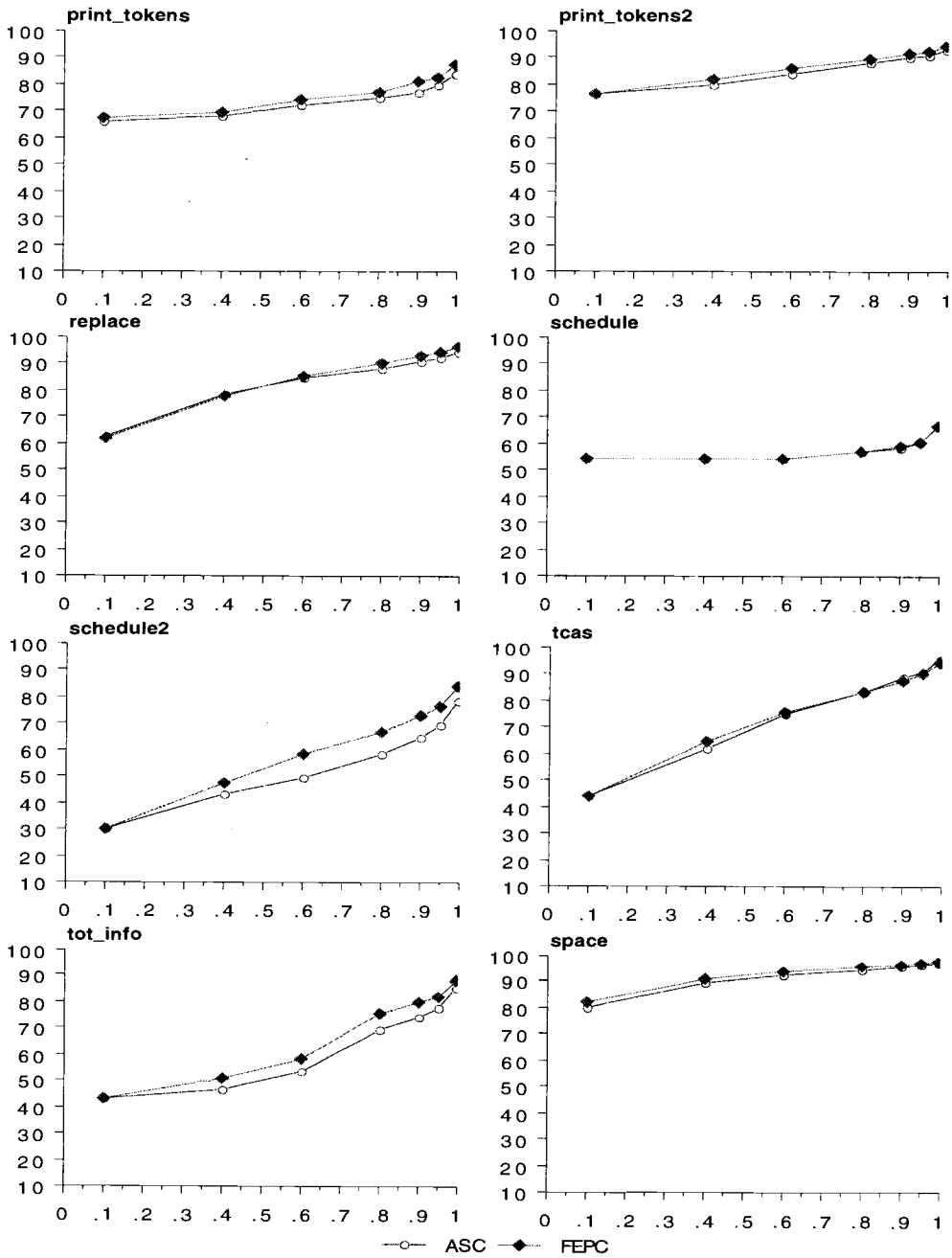

FIGURE 4.4: Average efficacy values of FEPC-adequate and ASC test suites, per program, run against the *Tough Fault Sets*. Efficacy is shown along the vertical axis and confidence level along the horizontal axis.

Program	*cl=0.1*		*cl=0.4*		*cl=0.6*		*cl=0.8*	
	mean dif.	ρ value	mean dif.	ρ value	mean dif.	ρ value	mean dif.	ρ value
print_tokens	**1.70**	**<.0001**	**1.40**	**<.0001**	**2.00**	**<.0001**	**2.10**	**<.0001**
print_tokens2	*0.03*	*.0799*	**1.60**	**<.0001**	**2.20**	**<.0001**	**1.50**	**<.0001**
replace	-0.90	<.0001	-0.50	<.0001	**1.00**	**<.0001**	**2.40**	**<.0001**
schedule	0.00	•	0.00	•	*0.04*	*.3299*	**0.30**	**<.0001**
schedule2	**0.20**	**<.0001**	**4.00**	**<.0001**	**9.10**	**<.0001**	**8.20**	**<.0001**
tcas	**0.40**	**<.0001**	**2.60**	**<.0001**	*0.30*	*.0629*	*0.30*	*.0752*
tot_info	0.00	•	**3.70**	**<.0001**	**4.70**	**<.0001**	**6.30**	**<.0001**
space	**2.40**	**<.0001**	**1.70**	**<.0001**	**1.20**	**<.0001**	**1.10**	**<.0001**
total	**0.50**	**<.0001**	**1.80**	**<.0001**	**2.60**	**<.0001**	**2.80**	**<.0001**

Program	*cl=0.9*		*cl=0.95*		*cl=0.995*		*total*	
	mean dif.	p value	mean dif.	p value	mean dif.	p value	mean dif.	p value
print_tokens	**4.00**	**<.0001**	**3.20**	**<.0001**	**3.80**	**<.0001**	**2.60**	**<.0001**
print_tokens2	**1.60**	**<.0001**	**1.20**	**<.0001**	**0.60**	**<.0001**	**1.20**	**<.0001**
replace	**2.30**	**<.0001**	**2.10**	**<.0001**	**1.50**	**<.0001**	**1.10**	**<.0001**
schedule	**0.10**	**.0158**	*0.03*	*.6146*	*0.01*	*.1613*	**0.10**	**<.0001**
schedule2	**8.00**	**<.0001**	**7.00**	**<.0001**	**5.60**	**<.0001**	**6.00**	**<.0001**
tcas	-0.13	<.0001	-0.40	.0008	-1.00	<.0001	**0.10**	**<.0001**
tot_info	**5.80**	**<.0001**	**4.30**	**<.0001**	**3.20**	**<.0001**	**4.00**	**<.0001**
space	**1.00**	**<.0001**	**1.00**	**<.0001**	**0.60**	**<.0001**	**1.30**	**<.0001**
total	**2.70**	**<.0001**	**2.30**	**<.0001**	**1.80**	**<.0001**	**2.10**	**<.0001**

TABLE 4.4: Results of paired t-test on data against *Tough Fault Sets.*

4.5.4 Test Suite Sizes

To address our third research question, regarding test suite sizes, Figure 4.5 contains eight graphs; each depicts test suite sizes for a program at all seven confidence levels. The individual boxplots show the distribution of test suite sizes at each confidence level.

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 became larger as confidence level increased. We performed regressions on the size data, and we found that quadratic polynomial curves can closely fit the data. The corresponding regression results are presented in Table 4.5. The high correlation coefficient values (R^2) provide firm evidence of polynormial growth trends in test suite size as confidence level increases.

Program	Regression Line	R^2
print_tokens	$y = 50.56 - 652.72 \times x + 2574.71 \times x^2 - 3738.58 \times x^3 + 1853.84 \times x^4$	0.967
print_tokens2	$y = 52.44 - 680.37 \times x + 2655.87 \times x^2 - 3804.59 \times x^3 + 1864.46 \times x^4$	0.967
replace	$y = 198.92 - 2825.64 \times x + 11235.08 \times x^2 - 16227.05 \times x^3 + 7972.49 \times x^4$	0.975
schedule	$y = 12.66 - 120.23 \times x + 467.06 \times x^2 - 680.02 \times x^3 + 337.35 \times x^4$	0.944
schedule2	$y = 120.38 - 1730.75 \times x + 6889.58 \times x^2 - 9928.10 \times x^3 + 4862.17 \times x^4$	0.975
tcas	$y = 97.31 - 1406.29 \times x + 5618.71 \times x^2 - 8122.93 \times x^3 + 3992.16 \times x^4$	0.975
tot_info	$y = 39.94 - 520.46 \times x + 2008.35 \times x^2 - 2855.98 \times x^3 + 1389.44 \times x^4$	0.945
space	$y = 1489.63 - 21232.12 \times x + 85514.95 \times x^2 - 123737.12 \times x^3 + 61136.91 \times x^4$	0.984

TABLE 4.5: Correlation between confidence level and test suite size, using polynomial regression.

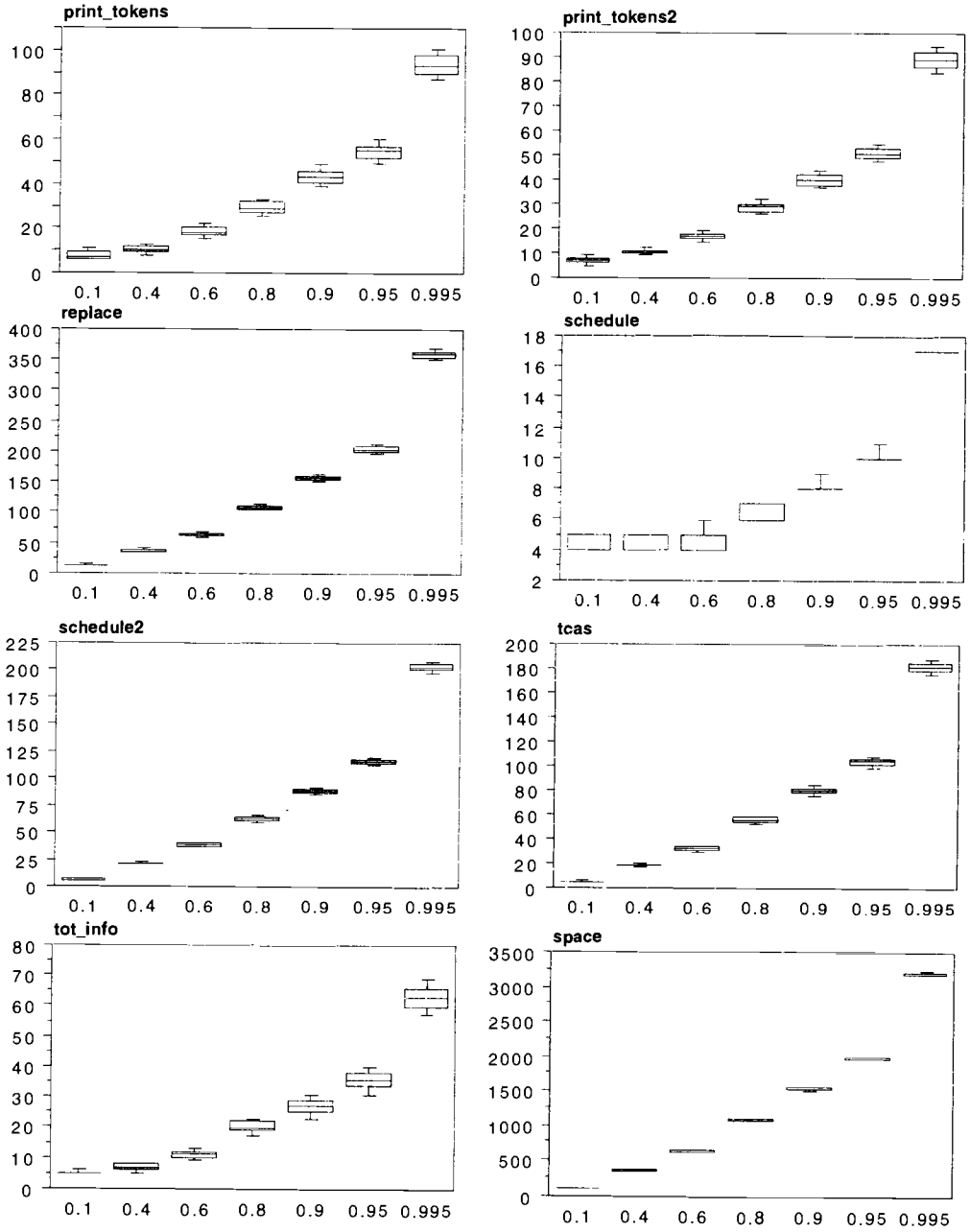

FIGURE 4.5: Boxplots of test suite sizes for eight programs.

4.6 Threats to Validity

We now discuss some of the threats to validity that exist for this study.

4.6.1 Threats to Internal 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. Since in this study, the FEPC-adequate and ASC test suites are generated by randomly selecting some test cases from the test pools, their fault-detection effectiveness is affected by this characteristic of the test pools. Further studies on this topic employing additional subject programs and test pools are necessary.

Second, our method for calculating FEP estimates provides one approach for approximating and using fault exposure probabilities, and these estimates may be inaccurate. However, one assumption underlying the FEPC adequacy test criterion is that we have correct FEP estimates so that we can transform them into correct hit number requirements. Inaccurate FEP estimates may result in inaccurate hit number requirements, and thus will affect the fault-detection effectiveness of FEPC-adequate test suites. We believe that more accurate approaches may exist, and Chapter 5 investigates an alternative.

4.6.2 Threats to External Validity

Threats to external validity limit our ability 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 to be 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. So, in future work, we should design some empirical studies to investigate the behavior of FEPC-adequate test suites under more complex error patterns. For example, we can design some faulty versions of a program by injecting several faults into each version at the same time.

4.6.3 Threats to Construct Validity

Threats to construct validity arise when measurement instruments do not adequately capture the concepts they are supposed to measure. There are four issues to consider. First, efficacy is not the only possible measure of test suite effectiveness. For example, our efficacy measure assigns 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. Third, this method of measuring fault-detection

effectiveness calculates effectiveness relative to a fixed set of faults, however in practice, during the testing procedure new faults may be introduced into the program when correcting those faults. Finally, our approach 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.

4.7 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 hypothesis 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 another 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.

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-adequate 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 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 the most interesting observation emerging from our results, however, concerns cost-benefits tradeoffs involving FEPC-adequate test suites. Consider the graph of efficacy results for `schedule2` in Figure 4.4. 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-coverage-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.

Chapter 5

IMPROVEMENT

5.1 Improvement on FEP Estimation

Our experiments showed that the incorporation of FEP estimates into the statement-coverage adequacy criterion could improve the fault-detection effectiveness of test suites. So, we wondered whether changes in our FEP estimation procedure can improve the fault-detection effectiveness of FEPC-adequate test suites.

In the method for calculating FEP estimates that we used in the previous empirical study, we treated all mutants as equal. However, through further study of the mutants we found that two types of mutants may need to be treated differently from other mutants; we call these *type-0 mutants* and *type-1 mutants*:

Type-0 mutants: Mutants that can not be killed by any test case in the test pool.

Type-1 mutants: Mutants that can be killed by every test case in the test pool that executes the corresponding statement.

As mentioned in Chapter 3, we can not easily determine whether a *type-0 mutant* is an *equivalent mutant* or not, so we did not simply eliminate them from our FEP estimate calculation. However, we can regard *type-0 mutants* as

representatives of a category of faults that are "extremely hard to detect". If a statement contains such a fault, no matter how many test cases we run though this statement, we can not guarantee that this fault will be detected.

In contrast to *type-0 mutants*, *type-1 mutants* can be detected by any test case in the test pool. So, no matter how many such faults exist in a statement, one test case that covers this statement is sufficient to detect all of them.

Therefore, we theorized that these two types of mutants are much less important than other types of mutants to the FEP estimate calculation procedure. We decided to treat them as two "equivalence classes" of mutants, such that for a given statement S, no matter how many mutants of S are in these two classes, one mutant per class is sufficient to represent each class in the FEP estimate calculation. Thus, we applied two adjustment rules to the original method of FEP estimate calculation:

adjustment 1: Let S be an executable statement in program P, such that S has m mutants, and n $(n > 0)$ of them are of *type-0*, then treat these n mutant as **one** *type-0 mutant* in FEP estimate calculation.

adjustment 2: Let S be an executable statement in program P, such that S has m mutants, and n $(n > 0)$ of them are of *type-1*, then treat these n mutant as **one** *type-1 mutant* in FEP estimate calculation.

For example: assume that statement S has 9 mutants, and there are 6 test cases in the test pool. Assume that the result of running these test cases on the mutants is as shown in Table 5.1. In this table, an entry of "1" means that the corresponding mutant can be killed by the corresponding test case, an entry of "0" means that the corresponding mutant can not be killed by the

	m1	m2	m3	m4	m5	m6	m7	m8	m9
t1	1	1	0	1	0	0	0	0	0
t2	1	1	1	0	1	0	0	0	0
t3	X	X	X	X	X	X	X	X	X
t4	1	1	1	0	1	0	0	0	0
t5	1	1	0	1	0	0	0	0	0
t6	1	1	0	1	0	0	0	0	0

TABLE 5.1: Results of test cases running over mutants of statement S.

corresponding test, and an entry of "X" means that the test case can't execute statement S.

From this table we can see that mutants $m1$ and $m2$ are *type-1 mutants*, so we count them only once. Mutants $m6 - m9$ are *type-0 mutants*, so they are also counted only once. So the number of "1" entries is: $5+2+3+2+0 = 12$, and the total number of entries is $5 + 3 \times 5 + 5 = 25$. The FEP estimate for statement S is $12/25 = 0.48$. If we used the old method, the FEP estimate for S would have been: $(5 \times 2 + 2 + 3 + 2)/(5 \times 9) \approx 0.38$.

5.2 Experiments with a New FEP Estimation Method

To determine whether this modified FEP estimation method can provide any improvement in the fault-detection effectiveness of FEPC-adequate test suites, we repeated the previous empirical study using this new FEP estimation method. The experiment results are presented in the following subsections.

5.2.1 *Results on* Original Fault Sets

Figures 5.1 and 5.2 depict the average test suite sizes and averge efficacy values of the paired FEPC-adequate and ASC adequate test suites measured against the *Original Fault Sets* over the seven confidence levels. These two figures contain eight graphs; each graph depicts the results for one subject program. These graphs are similar to the graphs presented in the figures in Chpater 4, except that to facilitate the comparison between the results of this experiment and the corresponding results of the previous experiment, plots from the previous experiment are included along with new plots.

From these graphs, we can see that under the new FEPC estimation method, `print_tokens` and `schedule2` exhibit larger gaps between the average efficacy values of FEPC-adequate test suites and those of ASC test suites. Programs `print_tokens2`, `schedule`, and `space` exhibit larger gaps too, although not as noticeably. On `replace`, in contrast to the results of the previous study, the new results show that the average efficacy values of FEPC-adequate test suites are greater than those of the ASC test suites. But `tcas` and `tot_info` exhibit smaller gaps in this experiment.

The graphs also indicate significant differences in average test suite sizes in the two experiments. For programs `print_tokens`, `schedule`, and `tot_info`, the average test suite sizes in this experiment are larger than in the previous study, while for the other five programs, average test suite sizes are smaller. On all programs other than `space`, the increase or decrease in the test suite size is accompanied with an increase or decrease in the average efficacy values of FEPC-adequate test suites, respectively. In contrast, `space` shows some small increase in average efficacy under confidence level 0.995, although the average

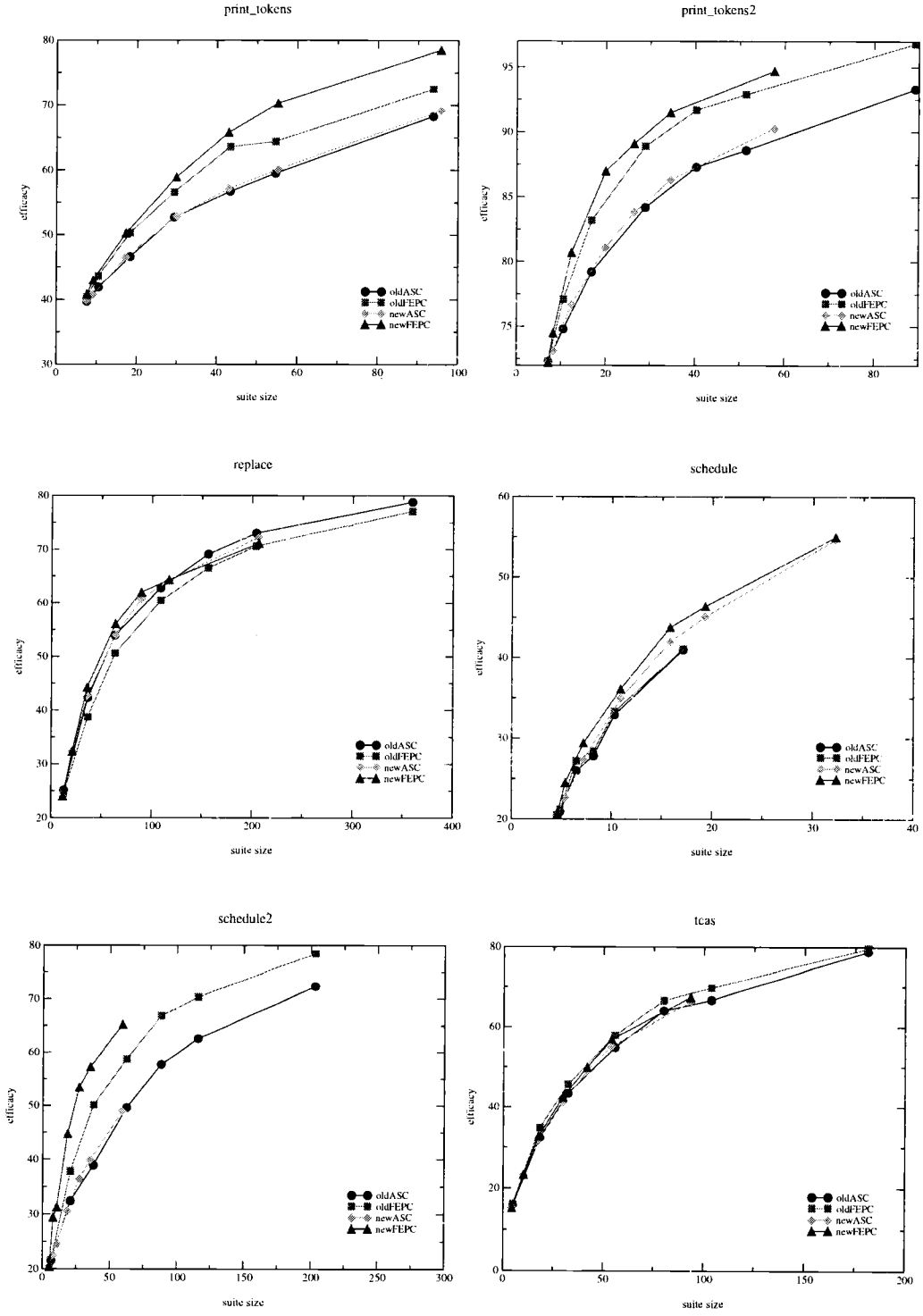

FIGURE 5.1: Results on *Original Fault Sets*.

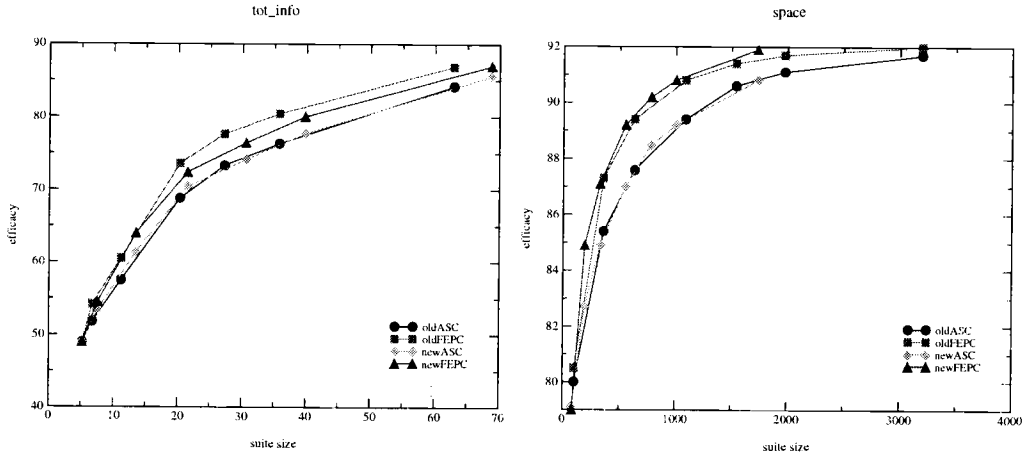

FIGURE 5.2: Results on *Original Fault Sets (cont.)*.

test suite size is almost only half of the average test suite size in the previous study.

As in the previous study, we ran paired t-tests on the paired efficacy values of FEPC-adquate and ASC test suites. Table 5.2 displays the paired t-test results on the 7000 efficacy measure pairs collected (across all seven confidence levels) for each program. The bottom row holds the t-test result on all 56,000 efficacy measure pairs. Also the corresponding data from the previous study is presented in this table, for purpose of comparison.

Judging by the mean difference values presented in the table, we can see that for six of the programs there is some improvement by applying the new FEP estimation method; but `tcas` and `tot_info` exhibit worse results. From the bottom row we can see that the new FEP estimation method has provided a 43% improvement in terms of mean difference in efficacy.

	old method		*new method*	
program	mean dif.	ρ value	mean dif.	ρ value
print_tokens	**3.70**	**<.0001**	**5.90**	**<.0001**
print_tokens2	**3.30**	**<.0001**	**3.70**	**<.0001**
replace	-2.30	<.0001	**0.60**	**<.0001**
schedule	**0.40**	**<.0001**	**1.20**	**<.0001**
schedule2	**7.00**	**<.0001**	**11.2**	**<.0001**
tcas	**2.10**	**<.0001**	**0.90**	**<.0001**
tot_info	**3.10**	**<.0001**	**1.70**	**<.0001**
space	**1.00**	**<.0001**	**1.50**	**<.0001**
total	**2.30**	**<.0001**	**3.30**	**<.0001**

TABLE 5.2: Results of paired t-tests on data against *Original Fault Sets*, for the old method and the new method.

5.2.2 Results on Mutation Fault Sets

Figures 5.3 and 5.4 depict average test suite sizes and average efficacy values measured against the *Mutation Fault Sets*. The corresponding data from the previous study are also presented in these two figures. We can see from the figures that `print_tokens`, `replace`, and `schedule2` display visible improvements in the fault-detection effectiveness of FEPC-adequate test suites, `tot_info` displays worse results, and in the graphs for the other four programs the average efficacy values are so close to each other that we can hardly tell the difference among them.

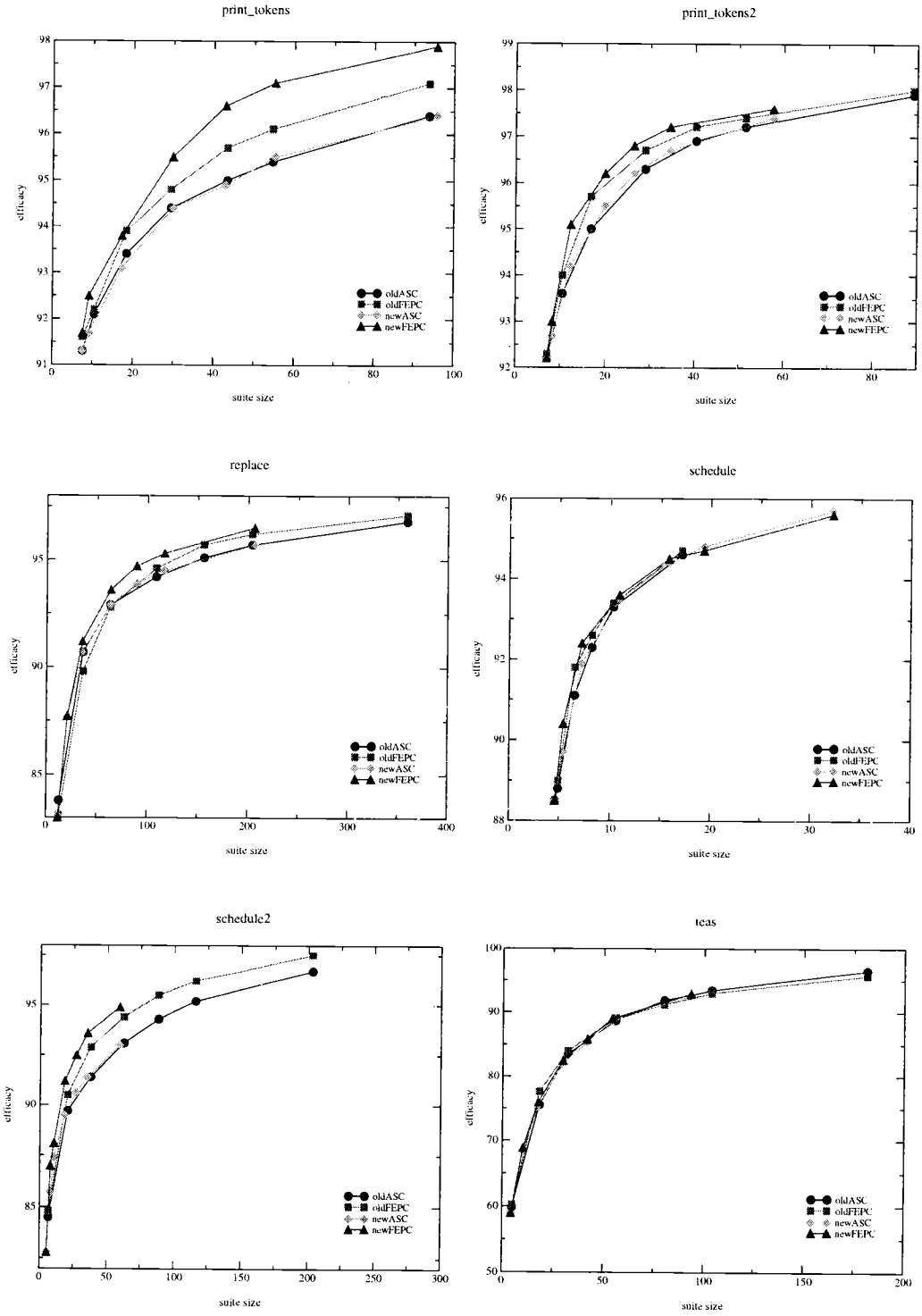

FIGURE 5.3: Results on *Mutation Fault Sets.*

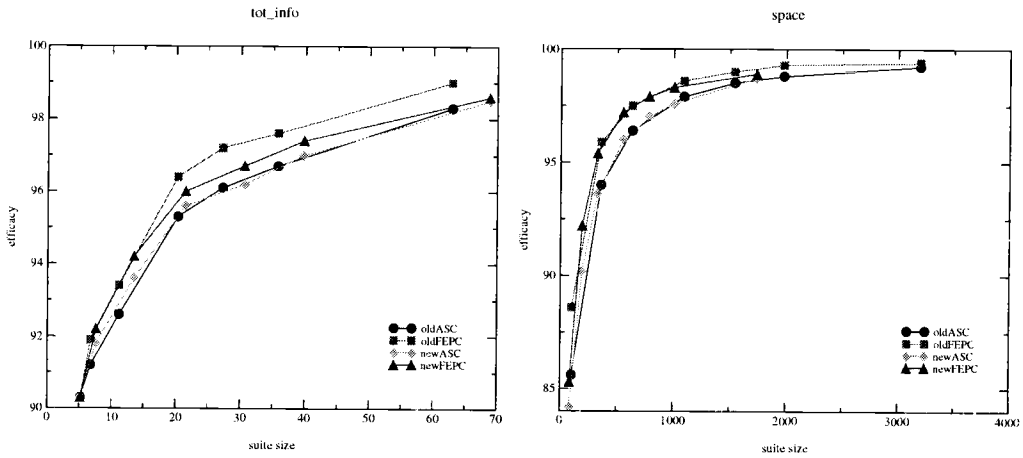

FIGURE 5.4: Results on *Mutation Fault Sets (cont.)*.

	old method		*new method*	
program	mean dif.	ρ value	mean dif.	ρ value
print_tokens	**0.40**	**<.0001**	**1.00**	**<.0001**
print_tokens2	**0.30**	**<.0001**	**0.40**	**<.0001**
replace	*0.09*	*.6010*	**0.40**	**<.0001**
schedule	**0.20**	**<.0001**	**0.20**	**<.0001**
schedule2	**0.90**	**<.0001**	**1.30**	**<.0001**
tcas	**0.20**	**<.0001**	**0.40**	**<.0001**
tot_info	**0.70**	**<.0001**	**0.30**	**<.0001**
space	**1.10**	**<.0001**	**1.10**	**<.0001**
total	**0.50**	**<.0001**	**0.60**	**<.0001**

TABLE 5.3: Results of paired t-tests on data against *Mutation Fault Sets*, for the old method and the new method.

The corresponding results of paired t-tests are presented in table 5.3. This table clearly shows that six of the programs display at least some improvement in fault-detection effectiveness of FEPC-adequate test suites under the new

FEP estimation method, while `space` exhibits identical results, and `tot_info` exhibits worse results. The bottom row indicates a total improvement of 20% mesured against the *Mutation Fault Sets*, which is much less significant than the improvement measured using the *Original Fault Sets*.

5.2.3 Results on Tough Fault Sets

The average test suite sizes and average efficacy values measuring against *Tough Fault Sets* are depicted in Figures 5.5 and 5.6, and the results of paired t-tests are presented in Table 5.4. The graphs in this figure are similar to the corresponding graphs for the results against the *Original Fault Sets*, but we can see differences in the results of the paired t-tests. This time, `space` supports the new FEP estimation method, and only `tot_info` exhibits negative results. The total improvement is about 33% in terms of mean difference in efficacy, less than the improvement seen with the *Original Fault Sets*, but better than that seen against *Mutation Fault Sets*.

5.2.4 Analysis of Results

Results of this experiment indicate that our modified FEP estimation method can yield some improvements in the fault-detection effectiveness of FEPC-adequate test suites, compared to the original estimation method. However, the program proves to be a factor that has great impact on this improvement: `print_tokens` and `schedule2` enjoy the greatest benefits from applying the new FEP estimation method, but `tot_info` suffers from the new method. Further study of this issue is needed.

Fault set is another factor that affects the results of this experiment. The

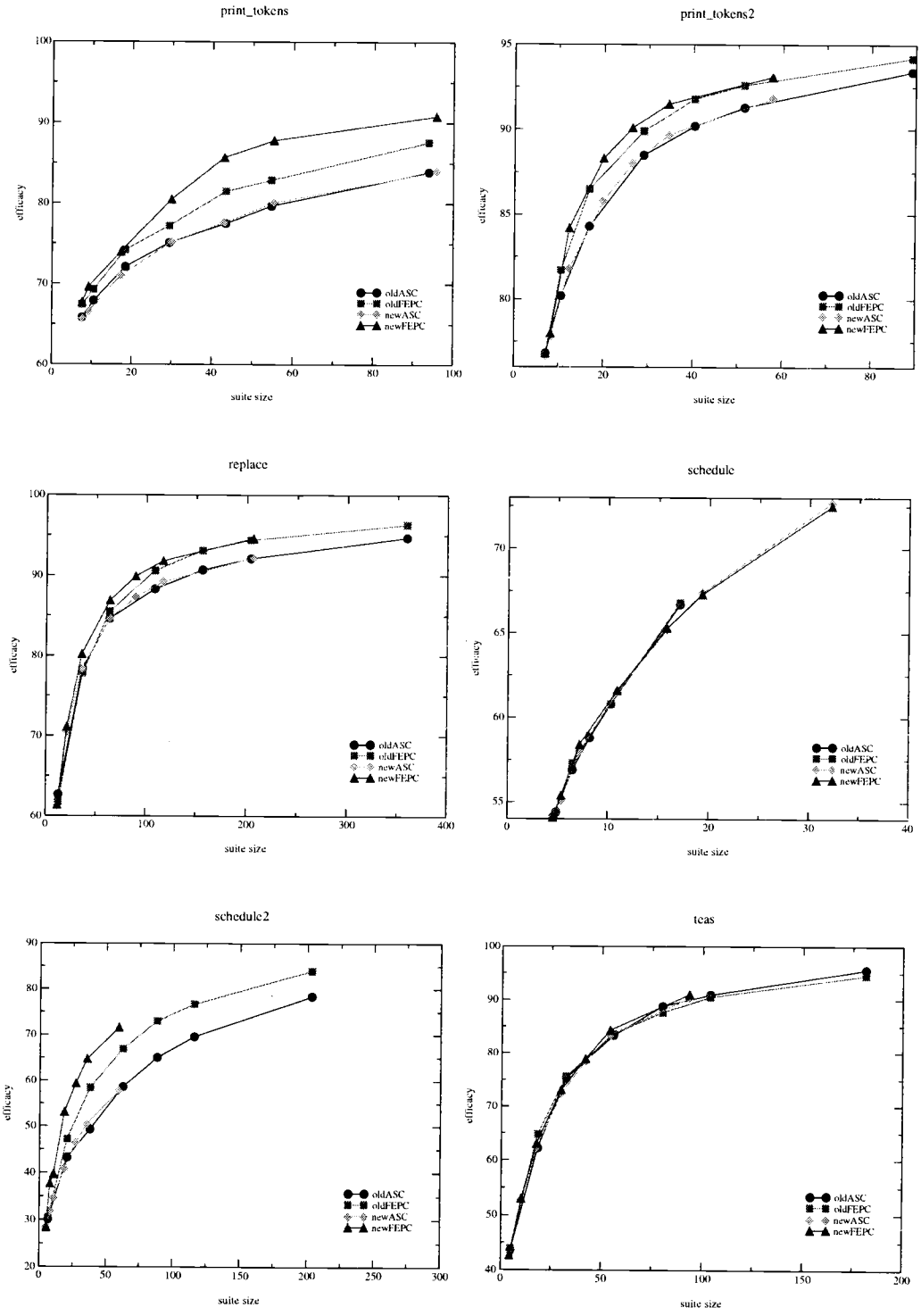

FIGURE 5.5: Results on *Tough Fault Sets.*

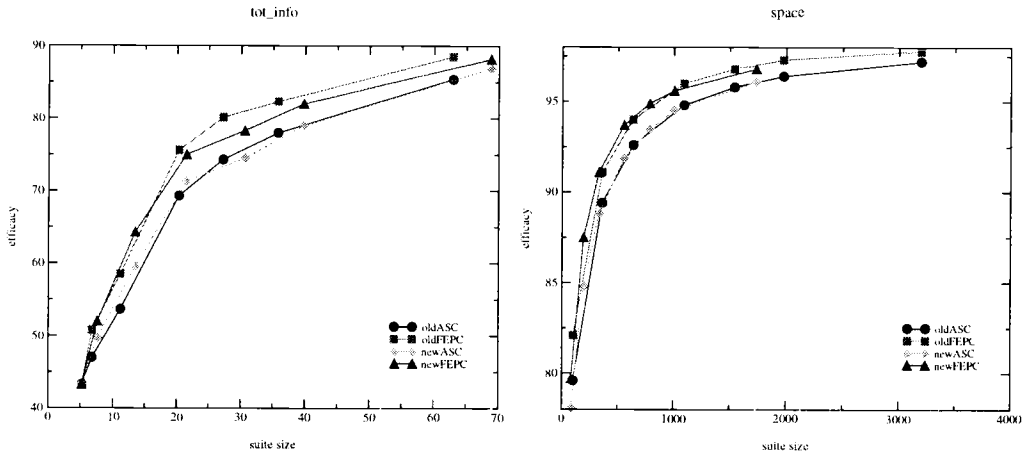

FIGURE 5.6: Results on *Tough Fault Sets (cont.)*.

	old method		*new method*	
program	mean dif.	ρ value	mean dif.	ρ value
print_tokens	**2.60**	**<.0001**	**5.20**	**<.0001**
print_tokens2	**1.20**	**<.0001**	**1.50**	**<.0001**
replace	**1.10**	**<.0001**	**1.80**	**<.0001**
schedule	**0.10**	**<.0001**	**0.10**	**.0276**
schedule2	**6.00**	**<.0001**	**9.20**	**<.0001**
tcas	**0.10**	**<.0001**	**0.60**	**<.0001**
tot_info	**4.00**	**<.0001**	**2.70**	**<.0001**
space	**1.30**	**<.0001**	**1.60**	**<.0001**
total	**2.10**	**<.0001**	**2.80**	**<.0001**

TABLE 5.4: Results of paired t-tests on data against *Tough Fault Sets*, for the old method and the new method.

improvements varied significantly on different fault sets. These results suggest that this new FEP estimation method, like the previous method, may be more useful for faults that are relatively hard to detect.

Chapter 6

CONCLUSIONS AND FUTURE WORK

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 series of formal experiments 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 experiments was not to evaluate a technique, but rather, to answer the important initial question of whether, if a cost-effective technique existed, it could be used to create cost-effective test suites.

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.

Our second study indicates that by altering our method for fault exposure probability estimation, we can obtain better fault-detection effectiveness. How-

ever, the effects of using the new method also varied significantly with factors such as program and fault type.

Our results provide some evidence supportive of FEP adequacy, however, before we can provide more definitive conclusions, further studies are necessary. Additional work can address following issues.

- As mentioned in Section 4.6, in order to reduce the threats to the validity of our studies, we require additional experiments, using more programs, different test pools, faulty versions with multiple faults, and so forth.

- As a feature of a code component, fault exposure potential has strong relationships with many other software features such as complexity, data dependencies, and control dependencies. How these factors affect the fault exposure potential of a code component needs to be carefully studied.

- In this work and other related work, the estimation of a code component's fault exposure potential is quantified as a single value. However, due to the diversity of the faults that may exist in a code component, we may explore the possibility of representing such potential by a range.

- We believe that there is room for improving our initial method of fault exposure potential estimation, because in that method we simply treat all mutants as equal and all test cases as equal. However, this treatment may not be realistic. By recognizing the differences among mutants and test cases we may be able to find ways to improve our estimation method. Our experiment reported in Chapter 5 provides an example of this. We hope that further studies can yield more significant improvements.

- Since mutation analysis is very expensive, we would like to find some way to reduce the cost of this procedure. Using constrained mutation analysis [15], which relies on a reduced set of mutants, may provide one approach.

- We can also explore some other ways to estimate fault exposure potential, such as Voas' PIE analysis [22], and Goradia's dynamic impact analysis [6].

- In the generation procedure for an FEPC-adequate test suite used in this work, a test case is added to a test suite if it can contribute some required hits to some statements. However, this test case may also contribute some unneeded hits to some other statements. So, inevitably, there will be some redundant test cases in a generated test suite: by executing this test suite many statements will have many more hits than they require. We would like to know whether reducing this redundancy in an FEPC-adequate test suite will affect its fault-detection effectiveness. There are two possible ways to reduce these redundant test cases in a test suite: to optimize the generation procedure or to minimize the already-existing FEPC-adequate test suites.

- A potential weakness of the FEPC-adequacy test criterion is that, although it may assign a high hit number requirement to a statement with low fault exposure potential, it does not choose test cases with any "intelligence". Such "blindness" in choosing test cases may hurt the effectiveness of the generated test suites. Thus, if there is some other information available that is useful for guiding test case selection, we could use it. This

suggests the possibility of combining FEPC adequacy test criteria with some other test adequacy criterion.

- In our experiments, the FEPC-adequate test criterion is investigated at the statement level. Since this test criterion is also applicable to other code components such as branches, paths, or data dependencies, we could investigate FEPC-adequacy test criteria based on these code components.

We hope that by such further research in this area, we can provide some cost-effective techniques for incorporating fault exposing potential estimates into testing. This work provides impetus for that research.

BIBLIOGRAPHY

[1] T. A. Budd, R. J. Demillo, and F. G. Sayward. The Design of a Prototype Mutation System fo Program Testing. In *Proc. ACM Nat. Comput. Conf.*, pages 623–627, 1978.

[2] 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, November 1989.

[3] Márcio E. Delamaro and José 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.

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

[5] Carlo Ghezzi, Mehdi Jazayeri, and Dino Mandrioli. *Fundamentals of Software Engineering.* Prentice Hall, Upper Saddle River, NJ, 1991.

[6] 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.

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

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

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

[10] W. E. Howden. Reliabillity of the path analysis testing strategy. *IEEE Trans. Softw. Eng.*, SE-2:208–215, September 1976.

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

[12] 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.

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

[14] Larry J. Morell. A theory of fault-based testing. *IEEE Transactions on Software Engineering*, 16(8):844–57, August 1990.

[15] 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, April 1996.

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

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

[18] D.J. Richardson and M.C. 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.

[19] G. Rothermel, M.J. Harrold, J. Ostrin, and C. Hong. An empirical study of the effects of minimization on the fault detection capabilities of test suites. In *Proceedings of Conference on Software Maintenance*, pages 34–43, November 1998.

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

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

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

[23] 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, November 1998.

[24] L.J. White. Software Testing and Verification. In *Advances in Computers.*, volume 26, pages 335–391, 1987.