Information and Software Technology 55 (2013) 897–917

Contents lists available at SciVerse ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

On the adoption of MC/DC and control-flow adequacy for a tight integration of program testing and statistical fault localization

Bo Jiang [a], Ke Zhai [b], W.K. Chan [c,*], T.H. Tse [b], Zhenyu Zhang [d]

[a] School of Computer Science and Engineering, Beihang University, Beijing, China
[b] Department of Computer Science, The University of Hong Kong, Pokfulam, Hong Kong
[c] Department of Computer Science, City University of Hong Kong, Tat Chee Avenue, Hong Kong
[d] State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China

ARTICLE INFO

Article history:
Available online 17 October 2012

Keywords:
Testing-debugging integration
Test case prioritization
Fault localization
Adequacy criterion
MC/DC

ABSTRACT

Context: Testing and debugging consume a significant portion of software development effort. Both processes are usually conducted independently despite their close relationship with each other. Test adequacy is vital for developers to assure that sufficient testing effort has been made, while finding all the faults in a program as soon as possible is equally important. A tight integration between testing and debugging activities is essential.

Objective: The paper aims at finding whether three factors, namely, the adequacy criterion to gauge a test suite, the size of a prioritized test suite, and the percentage of such a test suite used in fault localization, have significant impacts on integrating test case prioritization techniques with statistical fault localization techniques.

Method: We conduct a controlled experiment to investigate the effectiveness of applying adequate test suites to locate faults in a benchmark suite of seven Siemens programs and four real-life UNIX utility programs using three adequacy criteria, 16 test case prioritization techniques, and four statistical fault localization techniques. We measure the proportion of code needed to be examined in order to locate a fault as the effectiveness of statistical fault localization techniques. We also investigate the integration of test case prioritization and statistical fault localization with postmortem analysis.

Result: The main result shows that on average, it is more effective for a statistical fault localization technique to utilize the execution results of a MC/DC-adequate test suite than those of a branch-adequate test suite, and is in turn more effective to utilize the execution results of a branch-adequate test suite than those of a statement-adequate test suite. On the other hand, we find that none of the fault localization techniques studied can be sufficiently effective in suggesting fault-relevant statements that can fit easily into one debug window of a typical IDE.

Conclusion: We find that the adequacy criterion and the percentage of a prioritized test suite utilized are major factors affecting the effectiveness of statistical fault localization techniques. In our experiment, the adoption of a stronger adequacy criterion can lead to more effective integration of testing and debugging.

1. Introduction

Software testing can reveal program failures by running the program over a set of test cases. However, simply detecting program failures is inadequate. Developers must continue to debug the program, that is, to locate faults followed by fixing them. They also need to test the modified program to verify whether the identified fault has been removed as expected. Such an integrated loop of testing and debugging processes account for more than 30% of the total effort in typical software development [2]. Although testing and debugging techniques are under active research, a tight integration between testing and debugging is relatively understudied. A better understanding of such integration helps invent new methods that may further save project costs.

A fundamental problem in software testing is to know when to stop testing. Testers may apply a test adequacy criterion such as statement coverage to measure the progress of testing [1,5] and stop it accordingly. They may further use a test case prioritization technique [9,10,15,21,25] to reorder the test cases. As soon as the test adequacy criterion has been achieved, testers may stop applying further prioritized test cases even though the rest have been scheduled. Furthermore, if the executed test cases have exposed a program failure, testers may conduct debugging immediately.

* Corresponding author. Tel.: +852 3442 9684.
E-mail addresses: jiangbo@buaa.edu.cn (B. Jiang), kzhai@cs.hku.hk (K. Zhai), wkchan@cityu.edu.hk (W.K. Chan), thtse@cs.hku.hk (T.H. Tse), zhangzy@ios.ac.cn (Z. Zhang).

0950-5849/$ - see front matter © 2012 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.infsof.2012.10.001

In such situations, there are a few dimensions that affect which particular test cases are used for subsequent debugging. They include the kind of test adequacy criterion used, the kind of test case prioritization technique used, and the proportion of prioritized test cases used.

There are many forms of debugging techniques. A major category under active research in the last decade is statistical fault localization [3,12,22,23,27,31,32]. They are usually based on the coverage statistics of a set of executions to assess which particular program entity is more fault-relevant than other entities in the same program. Previous studies [16,23] have found empirically that the effectiveness of statistical fault localization to identify faults is heavily affected by the size of the test suite and the strategies used to prioritize test cases. On the other hand, the aspect of test adequacy criterion has not been studied.

To study the issue of integration between testing and debugging, we investigate in this paper the above three dimensions of test suite composition (namely, test adequacy criterion, test case prioritization technique, and the proportion of prioritized test cases used) from the perspective of statistical fault localization. Because the code coverage information on the program under regression test achieved by different prioritization strategies (such as random, adaptive random [15], meta-heuristics [21], and greedy [9]) on different types of adequate test suites can be different, statistical fault localization techniques are likely to demonstrate different effectiveness when using the corresponding execution statistics to locate fault-relevant program entities. Jiang et al. [16,17] studied the effectiveness of using the prioritized test suites produced by different test case prioritization strategies to locate faults by statistical fault localization techniques. In terms of the relative mean percentage of code examined to locate faults, they found empirically [17] that the random ordering and the additional strategy [9] can be more stable than the clustering-based and the total strategies. Nonetheless, they have not investigated the influence of the composition of adequate test suites that serve as inputs to test case prioritization and fault localization techniques. There are many important research questions remaining to be answered, such as the probability of obtaining a test suite that is both *adequate* with respect to some testing criterion and *effective* with respect to some fault localization technique, and whether the suggested suspicious region (that is, the list of statements suspected of containing a fault) can easily fit into a debug window on the canvas of a typical IDE, given that this suggestion is produced by a test suite that is deemed to be adequate.

In this paper, we report the results of an empirical study that applied three test adequacy criteria, 16 test case prioritization techniques, and four statistical fault localization techniques to 11 subject programs. We used Modified Condition/Decision Coverage (MC/DC) adequacy, branch adequacy, and statement adequacy because they are commonly practiced coverage criteria that can be applied to widely-used complex programs [1,5]. In total, we used 262 faulty program versions with 1000 different test suites for every adequacy criterion. Each prioritization technique was employed to prioritize each such test suite in every round of the best two test adequacy criteria (owing to our resource limitation in analyzing the huge amount of data). The corresponding execution statistics of the reordered test cases were utilized to locate faults using each of the four fault localization techniques. We also repeated the same procedure by systematically varying the proportion of each reordered test suite to simulate that only a portion of a reordered test suite can be used for fault localization. In total, we repeated such variations ten times for every test suite produced by each test case prioritization technique. As such, we have produced more than 330 million data points in terms of *Expense* [31]. It would be infeasible to report all the individual data in detail. Hence, we only report how well MC/DC adequacy, branch adequacy, and statement adequacy integrate with statistical fault localization and how they compare with one another at a summary level. We also studied whether the use of adequate test suites is more effective than random test suites in supporting fault localization.

Although many statistical fault localization research achievements have been obtained in the past decade, our empirical results still show many interesting findings. First, the use of MC/DC-adequate test suites is more effective in integrating with statistical fault localization than the use of branch-adequate test suites, which in turn is more effective than the use of statement-adequate test suites. Our result supports the conjecture that a stronger adequacy criterion supports statistical fault localization better. Second, the adoption of test case prioritization seems preferential, as supported by our result that shows no more than 30–40% of the test cases compromise the effectiveness of fault localization significantly in the statistical sense if the original test suites are adequate with respect to some test adequacy criterion. Last but not least, the result further shows that the fault localization techniques studied can still be ineffective in suggesting fault relevant statements that can easily fit into one debug window (e.g., 25 lines of code) in a typical IDE such as Eclipse. As such, we find that the current state of integration between testing and debugging techniques is still inadequate and far from satisfactory, which urges for more research.

This paper extends its conference version [14] in the following aspects:

(a) It reports the new result on MC/DC, a well-adopted adequacy criterion. It also reports the result of the use of the random strategy as a baseline "criterion" for comparison with the other three test adequacy criteria.
(b) It significantly extends the empirical study that includes a comprehensive study on how fault localization effectiveness in terms of the metric *Expense* varies against the change of size of test suites with respect to each test adequacy criterion.
(c) It proposes a metric *SavingRate*, which allows one to compare different adequacy criteria by normalizing an existing metric by the sizes of test suites.
(d) It analyzes our finding from the perspective of the subsumption relations among adequacy criteria in the integration of test case prioritization and statistical fault localization.

The main contribution of this paper with its preliminary studies [13,14] is as follows: (i) It presents the *first* controlled experiment to study the probability of obtaining a test suite that is both *adequate* with respect to a specific test adequacy criterion and *effective* with respect to a specific fault localization technique. (ii) It is the *first* study on the effectiveness of a hierarchy of adequacy criteria when their adequate test suites are utilized in statistical fault localization techniques. (iii) It proposes a *new* metric *SavingRate* that characterizes how effective an adequacy criterion is in the integration with statistical fault localization after discounting the impact of test suite sizes. (iv) It reports the *first* experimental results on how likely on average a test case prioritization technique may effectively integrate with a statement-level statistical fault localization technique if the original test suite is adequate with respect to a specific test adequacy criterion.

The rest of the paper is organized as follows: Section 2 reviews the test case prioritization and statistical fault localization techniques used in our study. We present our controlled experiment and its results in Section 3. Section 4 describes related work followed by a conclusion in Section 5.

2. Background

This section outlines the test case prioritization and statistical fault localization techniques applied in our empirical study.

2.1. Test case prioritization techniques

We follow [9] to categorize test case prioritization techniques in two dimensions. The first dimension is *granularity*, expressed in terms of program entities, which include *statements*, *branches*, and *functions* in the experiment. The second dimension is *prioritization strategy*. We study the *Greedy* [9] and the *ART* [15] strategies. The *Greedy* strategy can be further classified into the *Total* and *Additional* sub-strategies [9]. (We note that there are other types of greedy strategies, but they are not a part of our study, and hence we do not classify them here.) The *ART* strategy is reported in [15].

ART represents a strategy that randomly selects test cases followed by resolving the randomness among the selected test cases through a coverage measure. *Greedy* represents a strategy that selects test cases through a coverage measure followed by resolving tie cases randomly. As such, we refer to these two strategies as the *coverage-before-random* (*C2R*) *strategy* and the *random-before-coverage* (*R2C*) *strategy*, respectively. Table 1 summarizes the techniques studied in our experiment.

2.1.1. C2R strategy

When we pair up the two Greedy sub-strategies with the three levels of granularities, we produce six techniques: total statement (total-st), total branch (total-br), total function (total-fn), additional statement (addtl-st), additional branch (addtl-br), and additional function (addtl-fn). All of them have been reported in previous work [9]. The total statement (total-st) test case prioritization technique ranks test cases in descending order of the number of statements that they cover. When two test cases cover the same number of statements, it orders them randomly. The total branch (total-br) and the total function (total-fn) test case prioritization techniques are the same as total-st, except that they use branch coverage and function coverage information, respectively, instead of statement coverage information. The additional statement (addtl-st) test case prioritization technique is the same as total-st, except that in each round, it selects a test case that covers the maximum number of statements not yet exercised. When no remaining test case in the test suite can further improve the statement coverage, addtl-st resets all the statements to "not yet covered" and reapplies the same procedure to the set of remaining test cases. When two test cases cover the same number of additional statements in a round, it randomly picks one. The additional branch (addtl-br) and additional function (addtl-fn) test case prioritization techniques are the same as addtl-st, except that they use branch coverage and function coverage data, respectively, rather than statement coverage data.

2.1.2. R2C strategy

As mentioned above, we use the ART strategy [15] to represent the R2C strategy. The basic algorithm of ART reorders the test cases by iteratively constructing a candidate set of test cases, and then picks one test case out of the candidate set until all the test cases in a given regression test suite have been selected. To generate a candidate set of test cases, the algorithm randomly adds the not-yet-selected test cases one by one into the candidate set (which is initially empty) as long as they can increase the code coverage achieved by the candidate set. The algorithm then selects a test case from the candidate set that maximizes the distance of the test cases from the already selected test cases. The distance between two test cases is defined as the Jaccard distance between the coverage of the program entities of the two test cases. By combining three distance measures (average, minimum, and maximum) and the above three levels of granularities, we produce nine techniques: ART-st-maxmin, ART-st-maxavg, ART-st-maxmax, ART-fn-maxmin, ART-fn-maxavg, ART-fn-maxmax, ART-br-maxmin, ART-br-maxavg, and ART-br-maxmax. All the nine techniques have been defined and evaluated in our previous work [15].

2.2. Fault localization techniques

We revisit four statistical fault localization techniques used in our study. Each of them computes the suspiciousness of individual statements, followed by ranking these statements according to their suspiciousness scores. One of the techniques, namely *Tarantula* [19], further uses a tiebreaker to resolve statements having identical suspiciousness values so that the ranking can be fine-tuned. This set of techniques was also used in the experiment in previous work [29].

Table 2 summarizes these fault localization techniques. The function *%failed*(*s*) in the table is the percentage of failed test cases that execute a statement *s* (among all the failed test cases in the

Table 1
Test case prioritization techniques.

Ref.	Acronym	Brief description	
*T*1	Random	Random ordering	
Greedy (C2R)			
*T*2	total-st	Total statement	
*T*3	total-fn	Total function	
*T*4	total-br	Total branch	
*T*5	addtl-st	Additional statement	
*T*6	addtl-fn	Additional function	
*T*7	addtl-br	Additional branch	
		Level of coverage	Test set distance
ART (R2C)			
*T*8	ART-st-maxmin	Statement	Maximize the minimum distance between test cases
*T*9	ART-st-maxavg		Maximize the average distance between test cases
*T*10	ART-st-maxmax		Maximize the maximum distance between test cases
*T*11	ART-fn-maxmin	Function	Maximize the minimum distance between test cases
*T*12	ART-fn-maxavg		Maximize the average distance between test cases
*T*13	ART-fn-maxmax		Maximize the maximum distance between test cases
*T*14	ART-br-maxmin	Branch	Maximize the minimum distance between test cases
*T*15	ART-br-maxavg		Maximize the average distance between test cases
*T*16	ART-br-maxmax		Maximize the maximum distance between test cases

Table 2
Statistical fault localization techniques.

Technique	Ranking formula
Tarantula [19]	$\frac{\%failed(s)}{\%failed(s)+\%passed(s)}$
	Tie-breaker
	max ($\%failed(s)$, $\%passed(s)$)
Adapted Statistical Bug Isolation (SBI) [22]	$\frac{failed(s)}{passed(s)+failed(s)}$
Jaccard [3]	$\frac{failed(s)}{totalfailed+passed(s)}$
Ochiai [3]	$\frac{failed(s)}{\sqrt{total(failed\times(s)+passed(s))}}$

test suite). The function $\%passed(s)$ is the percentage of passed test cases that execute that statement. The functions $failed(s)$ and $passed(s)$ calculate the number of failed and passed test cases, respectively, that exercises statement s. The variable *totalfailed* is the total number of failed test cases.

3. Controlled experiment

In this section, we report on our controlled experiment.

3.1. Research questions

We raised three new and important research questions to study the issue of integrating testing and debugging in relation to test adequacy.

- **RQ1**: How likely is it that a test suite produced to satisfy a test adequacy criterion is effective for fault localization?
- **RQ2**: How do different test adequacy criteria compare with one another after factoring out the impact of the test suite size?
- **RQ3**: After prioritizing the test cases in an effective test suite, what minimum portion of this test suite can have the same fault localization effectiveness as the whole adequate suite?

From the study of RQ1, developers and researchers will have a better understanding of the probability of generating effective test suites based on test adequacy criteria with respect to some of the best and representative statistical fault localization techniques. Having a high probability makes developers more comfortable in employing such adequate test suites to perform regression testing on their programs and using the execution statistics to aid fault localization activities. On the other hand, if the probability is found to be low, the result can alert developers when using such test suites. They may consider improving the test suites before use to support their testing and debugging activities. For RQ1, we used test suites of various sizes that satisfy a specific testing adequacy criterion. This is because we want to evaluate the "absolute" fault localization effectiveness of the test suites generated by different adequacy criteria. Thus, evaluating RQ1 without taking test suite size into consideration helps us better understand the fault localization capability of adequate test suites in practice.

Answering RQ2 will enable us to gauge the relative fault localization effectiveness of the test suites generated by different test adequacy criteria. Different adequacy criteria will inherently require different test suite sizes, which is a known factor affecting fault localization effectiveness. Adding more test cases to a smaller adequate test suite will introduce redundancy with respect to the test adequacy criterion used to produce the test suite, which makes the assessment of test adequacy criteria not meaningful. Similarly, removing some test cases from a larger adequate test suite will make the latter inadequate with respect to the test adequacy criterion. Again, by so doing, it makes the assessment of test adequacy criteria not meaningful. As such, in the data analysis, we normalize the test suite sizes to make the comparison among different test adequacy criteria fairer.

Answering RQ3 helps developers and researchers decide whether the effort on prioritizing test cases is worthwhile and whether executing only the higher priority portion of the prioritized test cases still retains good fault localization effectiveness. If the finding is positive, developers may be comfortable in using a portion of test data for fault localization. On the other hand, if the finding is negative, additional test cases must be used to prevent the fault localization effectiveness of the test suites from being seriously compromised.

3.2. Subject programs

Our experiments used the Siemens suite and four UNIX programs as subjects, as shown in Table 3. For each subject program, the table shows the name, the number of faulty versions, the executable lines of code, the test pool size, and the average percentage of compound Boolean expressions in decisions in relation to all Boolean expressions in decisions. It is the existence of compound Boolean expressions in the decision statements in a program that makes the MC/DC adequacy criterion different from the branch adequacy criterion. All the subject programs were downloaded from the Software-artifact Infrastructure Repository (SIR) [7]. Each subject came with, among other files, a set of faulty versions and a test pool. In this work, we use single fault versions of the subject programs to study the proposed research questions.

3.3. Test adequacy criteria

We used three adequacy criteria, namely, statement adequacy, branch adequacy, and MC/DC adequacy. We chose *branch coverage* and *statement coverage* because they are commonly used criteria that are widely applicable to industrial-strength programs [5]. Moreover, many existing industrial-strength tools (such as gcov) can provide profiling data for testers to determine whether these

Table 3
Subject programs.

Group	Subject	No. of faulty versions	LOC	Test pool size	Percentage of compound Boolean
Siemens suite	tcas	41	133–137	1608	2.4
	schedule	9	291–294	2650	3.2
	schedule2	10	261–263	2710	1.0
	tot_info	23	272–274	1052	5.6
	print_tokens	7	341–342	4130	1.7
	print_tokens2	10	350–354	4115	5.4
	replace	32	508–515	5542	2.0
UNIX programs	flex (2.4.7–2.5.4)	21	8571–10124	567	5.5
	grep (2.2–2.4.2)	17	8053–9089	809	14.1
	gzip (1.1.2–1.3)	55	4081–5159	217	11.6
	sed (1.18–3.02)	17	4756–9289	370	11.6

two coverage criteria have been achieved. We also chose the MC/DC adequacy criterion because it is a well-adopted adequacy criterion in the aeronautics industry. For instance, MC/DC is used in the FAA's DO-178 standard [1] to ensure that the most safety-critical software is tested adequately. Thus, a study of the three adequacy criteria can provide valuable contribution to the advance in the state of the art in research and the advance in the state of the practice in industry.

The *statement* (or *branch*, respectively) adequacy criterion ensures that each statement (or branch, respectively) is covered at least once by a test suite. The *Modified Condition/Decision Coverage* (*MC/DC*) adequacy criterion [1] requires a number of conditions: (i) for each decision statement *d*, every point of entry or exit should be tested at least once, (ii) every condition in *d* has taken all possible outcomes at least once, (iii) every decision statement in the program has taken all possible outcomes at least once, and (iv) each condition in *d* has been shown to independently affect the outcome of *d*. A condition is shown to independently affect the outcome of a decision by varying only that condition while all the other possible conditions remain unchanged. To make the study complete, we also included the random strategy as a baseline criterion for comparison.

For the Siemens programs, SIR provides a set of 1000 branch-adequate test suites, 1000 statement-adequate test suites, and 1000 random test suites. We further constructed 1000 MC/DC-adequate test suites using the test pool for each Siemens program. However, only one test pool is available for each UNIX program. We used this test pool to construct 1000 branch-adequate test suites, 1000 statement-adequate test suites, and 1000 MC/DC-adequate test suites for each UNIX program. The test suites are constructed independently of one another, while a test case may belong to more than one test suite. The sizes of the test suites for all programs range from 16 to 289. To generate a MC/DC-adequate test suite, we iteratively selected test cases from the test pool and added them into the test suite if the coverage achieved by the constructed test suite could be improved in terms of the criterion. The means and standard deviations of the levels of adequacies achieved by different adequacy suites generated in our experiment are shown in Table 4. For example, the mean percentage of statement coverage achieved by all the generated suites for tcas is 95% with a standard deviation 5%. When generating random test suites for the UNIX programs, we simply selected test cases randomly from the test pool until the required suite size has been reached. We randomly picked a value as the size of each random test suite to be constructed, and produced 1000 random test suites.

3.4. Metrics

3.4.1. Expense

To measure the effectiveness of fault localization, we used the *Expense* metric [12] defined as:

$$\text{Expense} = \frac{\text{rank of faulty statement}}{\text{total number executable statement}}$$

where the rank of a given statement is the sum of (a) the number of statements with higher suspiciousness values and (b) the number of statements with identical suspiciousness values and identical or higher tiebreaker values.

3.4.2. FLSP

In practice, a developer may only walk through a small portion of the source code. As a result, a high value of *Expense* (such as 90%) may be useless for debugging. A sequence of test cases with respect to a fault localization technique and a given faulty program is said to be *α-effective* if the value of *Expense* using this sequence of test cases by the fault localization technique on the faulty program is strictly lower than the threshold value specified by α.

If this is the case, we say that "the test suite *supports* the fault localization technique". As such, the proportion of adequate test suites for a test adequacy criterion *C* empirically represents the probability that a *C*-adequate test suite supports a statistical fault localization technique *T*. We use this probability to measure how well *C* supports *T*.

Given a faulty program and a fault localization technique *T*, we define the metric *Fault Localization Success Percentage* (*FLSP*) as the ratio between the number of α-effective test suites in a test suite pool *P* and the size of *P* (denoted by $|P|$), thus:

$$FLSP(T,\ P,\ \alpha) = \frac{\{|t \in\ P|t \text{ is } \alpha\text{-effective }|\}}{|P|}$$

3.4.3. SavingRate

We define *Saving* as the percentage of code that need not be examined to locate a fault.

$$Saving = 1 - Expense$$

Thus, the higher the value of *Saving*, the better will be the fault localization result.

However, when discussing the impact of an adequate test suite on statistical fault localization, an important factor to be considered is the number of test cases in the suite. Generally speaking, if a test adequacy criterion *A* subsumes another criterion *B*, satisfying criterion *A* will require more (if not the same) test cases than satisfying criterion *B*. The number of test cases within a test suite is, however, a confounding factor in comparing fault localization effectiveness achieved by different test suites. To compare the effectiveness of different test adequacy criteria in a more objective manner, it is important to control the impact of the test suite size.

We propose a derived metric *SavingRate* to measure the average effectiveness of a testing criterion in supporting statistical fault localization. Given a test suite of size *n*, we define *SavingRate* by the formula:

Table 4
Levels of adequacies achieved for the subject programs.

Subject	Statement		Branch		MC/DC	
	Mean (%)	Standard deviation (%)	Mean (%)	Standard deviation (%)	Mean (%)	Standard deviation (%)
Tcas	95	5	96	5	95	3
schedule	96	4	97	5	94	6
schedule2	96	6	95	8	95	7
tot_info	98	6	98	3	97	4
print_tokens	99	5	99	2	95	6
print_tokens2	99	3	97	4	95	5
replace	98	5	98	3	94	6
flex 2.4.7–2.5.4	97	6	96	8	98	4
grep 2.2–2.4.2	95	3	97	6	93	3
gzip 1.1.2–1.3	98	3	98	4	96	4

$$SavingRate = \frac{saving}{n}$$

Intuitively, *SavingRate* is a measure of the fault localization effectiveness per test case. It is a measure of the fault localization capability of different adequacy criteria normalized against test suite size.

3.5. Experimental setup

We applied each test case prioritization technique (see Table 1) and each fault localization technique (see Table 2) to every adequate test suite of every subject program. For every prioritized test suite generated by each test case prioritization technique, we repeated the above procedure using, in turn, the top 10%, 20%,..., 90% of the prioritized test suite. For every such portion of all the prioritized test suites applicable to every corresponding version of each subject program, we collected the values of *Expense* for each fault localization technique, and computed the FLSP values and the *SavingRate* values.

We conducted the experiment on a Dell PowerEdge 2950 server serving a Solaris UNIX system. We used gcc version 4.4.1 as the C compiler. The server has two Xeon 5430 (2.66 GHz, 4 core) processors and 4 GB physical memory. We followed previous test case prioritization studies [31] to remove faulty versions that cannot be detected by any test case in the test pool as well as those that can be detected by more than 20% of the test cases in the pool. We used the gcov tool with the gcc compiler to collect the execution statistics for each execution.

To study RQ1 and RQ2, we used all the random, branch-adequate, statement-adequate, and MC/DC-adequate test suites for our experimentation. For each faulty version, we also removed those test suites that cannot detect the fault because the fault localization techniques we used require at least one failed test case. We also removed all the test suites whose results on our platform differed from those indicated in the downloaded benchmark. We then passed the execution statistics to all the four fault localization techniques. For RQ1, we followed [13] to measure their results in terms of FLSP on all subject programs with three different fault localization effectiveness threshold values (1%, 5%, and 10%). For RQ2, we calculated the *SavingRate* for all test suites generated from all adequacy criteria and all subject programs.

We studied RQ1 and RQ2 separately rather than merging them because they are targeting at different goals. For RQ1, we used the test suites satisfying a specific testing adequacy criterion without considering test suite size. In this way, we can evaluate the "absolute" fault localization effectiveness of the test suites generated by different adequacy criteria in practice. For RQ2, we are interested in comparing different adequacy criteria while controlling the confounding factor of test suite size. This makes the comparison between different adequacy criteria fairer.

RQ3 is a follow-up research question based on the results of RQ1 and RQ2. We used the two best adequacy criteria from the two previous research questions, namely, branch adequacy and MC/DC adequacy. In this way, we can still preserve the generality of our findings while controlling the scale of our empirical study. Similarly to RQ1 and RQ2, we removed all the test suites that contain no failed test case as well as all test suites that cannot work on our platform.

All ART techniques are adapted from random selection. We followed [13] to repeat each of them 20 times so as to obtain an average performance and to select 50 suites out of the available 1000 test suites for every Siemens and UNIX subject program. Thus, we conducted a total of 1000 prioritizations for every ART technique on each subject. We then used MATLAB for ANOVA tests and multiple comparisons on the mean values, and specified a 5% significance level for hypothesis testing.

3.6. Data analysis

3.6.1. Answering RQ1

We studied the effectiveness of a fault localization technique using all the test cases within an adequate or random test suite. As a result, we need not distinguish between different test case prioritization techniques, as the test suites generated by them will have the same fault localization results. Tables 5–12 present the means and standard deviations of the numbers of effective suites averaged over all faulty versions of the Siemens and UNIX programs on Tarantula, SBI, Jaccard, and Ochiai, respectively. In each table, the first row shows the threshold values used in the experiment. We used three threshold values to determine the effectiveness of fault localization, namely, 1%, 5%, and 10%. In other words, if a fault can be located by inspecting less than 1% (or 5% or 10%) of the suggested list of suspicious statements, we deem the fault localization result to be effective. Based on the threshold values, we obtained three such groups of results. The second row shows four adequacy criteria for each group, namely, Rand for the random strategy, Stmt for statement adequacy, Br for branch adequacy, and MC/DC for MC/DC adequacy. The remaining rows show the means and standard deviations of the numbers of effective test suites for each program. (The total number of test suites for each faulty version is 1000.)

We studied how different adequacy criteria compare with one another without considering test suite size. We observe from Table 5 that for every subject program and every threshold value, on average, the use of an MC/DC-adequate test suite performs consistently better than the use of a branch-adequate test suite, which in turn performs consistently better than the use of a statement-adequate test suite. We further conducted ANOVA analysis between MC/DC- and branch-adequate test suites as well as between branch- and statement-adequate test suites to see whether each

Table 5
Mean numbers of effective test suites for Tarantula.

Threshold value	$\alpha = 1\%$				$\alpha = 5\%$				$\alpha = 10\%$			
Adequacy criteria	Rand	Stmt	Br	MC/DC	Rand	Stmt	Br	MC/DC	Rand	Stmt	Br	MC/DC
tcas	5	8	25	34	23	36	191	221	23	31	193	212
replace	32	46	145	166	77	126	342	363	107	140	381	399
tot_info	59	74	145	160	82	140	331	349	102	181	430	470
schedule	0	1	14	18	37	52	169	198	45	61	243	281
schedule2	0	0	0	0	4	5	35	45	9	11	107	123
print_tokens	1	3	55	59	17	22	151	178	21	37	215	247
print_tokens2	18	56	156	163	83	121	317	361	101	159	332	387
grep	121	371	618	719	348	546	832	867	432	756	936	962
sed	108	319	604	732	401	594	860	886	511	753	931	971
flex	124	376	665	750	335	586	827	890	486	720	951	979
gzip	131	395	674	681	362	516	843	832	502	711	945	978

Table 6
Standard deviations of the numbers of effective test suites for Tarantula.

Threshold value	$\alpha = 1\%$				$\alpha = 5\%$				$\alpha = 10\%$			
Adequacy criteria	Rand	Stmt	Br	MC/DC	Rand	Stmt	Br	MC/DC	Rand	Stmt	Br	MC/DC
tcas	2	4	3	5	6	0	4	5	4	3	6	4
replace	6	7	6	7	5	5	6	7	10	8	6	4
tot_info	4	5	8	9	13	3	7	9	5	7	8	6
schedule	0	0	3	2	3	5	4	5	4	4	3	3
schedule2	0	0	0	0	0	0	2	3	2	2	3	4
print_tokens	0	1	4	6	3	3	1	4	3	4	6	4
print_tokens2	5	6	5	3	7	8	7	6	5	5	2	5
grep	30	26	28	20	25	21	26	28	27	24	22	32
sed	22	27	22	23	26	29	23	29	30	27	23	33
flex	25	31	19	18	28	33	32	34	16	18	12	18
gzip	26	33	31	31	28	30	34	26	21	19	10	12

Table 7
Mean numbers of effective test suites for SBI.

Threshold value	$\alpha = 1\%$				$\alpha = 5\%$				$\alpha = 10\%$			
Adequacy criteria	Rand	Stmt	Br	MC/DC	Rand	Stmt	Br	MC/DC	Rand	Stmt	Br	MC/DC
tcas	7	11	29	29	29	46	205	232	25	39	201	205
replace	36	48	152	172	86	134	353	357	106	154	388	396
tot_info	71	80	157	166	94	141	335	348	117	186	437	463
schedule	6	10	19	21	39	63	180	203	46	70	246	273
schedule2	0	0	0	0	10	15	38	43	10	16	109	114
print_tokens	3	5	61	66	24	35	153	165	30	43	215	239
print_tokens2	45	70	166	179	87	132	317	336	104	168	337	361
grep	193	306	690	731	393	595	831	889	445	742	952	958
sed	225	389	681	763	401	590	880	898	496	719	946	963
flex	186	306	678	719	335	541	860	903	498	754	940	961
gzip	205	324	611	648	362	548	818	843	531	759	953	972

Table 8
Standard deviations of the numbers of effective test suites for SBI.

Threshold value	$\alpha = 1\%$				$\alpha = 5\%$				$\alpha = 10\%$			
Adequacy criteria	Rand	Stmt	Br	MC/DC	Rand	Stmt	Br	MC/DC	Rand	Stmt	Br	MC/DC
tcas	2	2	3	4	5	6	4	7	4	5	7	6
replace	5	3	6	3	4	5	6	5	6	7	5	6
tot_info	3	5	8	5	6	5	4	13	16	14	13	21
schedule	0	0	0	2	0	0	0	0	3	5	6	3
schedule2	0	0	0	0	0	0	0	0	0	2	3	7
print_tokens	0	0	3	5	4	3	4	6	3	5	4	5
print_tokens2	5	6	3	5	6	5	5	6	7	5	6	4
grep	16	18	12	13	14	12	15	14	13	16	18	17
sed	23	30	22	24	20	18	25	20	16	18	11	20
flex	13	19	24	20	16	19	20	22	15	20	16	19
gzip	22	19	17	18	19	21	22	25	16	11	15	14

pair differs significantly. The results give small p-values (0.0034 and 0.0049, respectively), which successfully reject the null hypothesis that there is no difference between each pair of them and confirms our observation at a 5% significance level.

From Table 6, we observe that the standard deviations of the numbers of effective test suites for different adequacy criteria are comparable to one another. We further conducted hypothesis testing, which produced a large p-value (0.26) to confirm our observation.

Moreover, the results from Tables 5 and 6 are consistent with those of the other three fault localization techniques in Tables 7–12. The relative order is also consistent with the subsumption relationships among the three test adequacy criteria, where MC/DC adequacy subsumes branch adequacy, which in turn subsumes statement adequacy (in terms of test coverage requirements). Furthermore, the use of each adequacy criterion in supporting statistical fault localization is more effective than random testing. The result is consistent with our conjecture that the adoption of adequacy criteria is, in general, worthwhile.

We also observe that the probability of obtaining an effective test suite for a UNIX program is significantly higher than that for a program in Siemens suites. This observation is interesting. Future work should study the underlying rationales.

To have a better understanding of the results, Fig. 1 shows all the data points in the experiment, each of which represents the code examination effort to locate a fault (measured by *Expense*) for program versions of different sizes (measured by lines of code). The x-axis shows the lines of code for all the Siemens and UNIX programs while the y-axis shows the number of lines in the source code that need to be examined to find a fault. Each dot represents

Table 9
Mean numbers of effective test suites for Jaccard.

Threshold value	$\alpha = 1\%$				$\alpha = 5\%$				$\alpha = 10\%$			
Adequacy criteria	Rand	Stmt	Br	MC/DC	Rand	Stmt	Br	MC/DC	Rand	Stmt	Br	MC/DC
tcas	6	9	31	38	28	41	235	245	25	37	217	232
replace	17	24	42	46	25	36	206	215	20	33	207	235
tot_info	35	56	153	162	84	138	354	362	103	154	394	421
schedule	51	80	147	155	107	158	338	359	126	186	433	456
schedule2	8	13	31	40	43	69	180	192	46	69	256	306
print_tokens	0	0	0	0	8	11	36	42	7	11	109	153
print_tokens2	11	16	73	82	20	32	156	181	35	51	216	230
grep	211	306	693	713	343	512	844	867	474	687	920	970
sed	272	389	660	695	333	513	799	812	493	747	885	899
flex	193	306	685	692	352	510	805	822	494	705	903	937
gzip	220	324	670	711	369	568	852	890	480	695	915	928

Table 10
Standard deviations of the numbers of effective test suites for Jaccard.

Threshold value	$\alpha = 1\%$				$\alpha = 5\%$				$\alpha = 10\%$			
Adequacy criteria	Rand	Stmt	Br	MC/DC	Rand	Stmt	Br	MC/DC	Rand	Stmt	Br	MC/DC
tcas	0	0	2	6	7	6	6	10	2	12	5	8
replace	4	4	8	6	6	12	8	10	3	10	9	11
tot_info	6	5	7	5	6	8	7	8	7	6	11	14
schedule	2	4	5	4	12	11	9	10	8	9	6	7
schedule2	5	11	7	6	6	8	7	9	5	9	6	6
print_tokens	0	0	0	0	0	0	3	2	2	4	10	9
print_tokens2	2	0	4	3	6	3	4	4	12	10	9	8
grep	14	13	9	11	10	16	12	26	21	11	13	18
sed	16	17	21	23	19	25	13	27	24	8	10	5
flex	12	16	19	15	16	18	22	24	22	15	25	16
gzip	11	22	25	22	24	18	25	22	19	20	32	27

Table 11
Mean numbers of effective test suites for Ochiai.

Threshold value	$\alpha = 1\%$				$\alpha = 5\%$				$\alpha = 10\%$			
Adequacy criteria	Rand	Stmt	Br	MC/DC	Rand	Stmt	Br	MC/DC	Rand	Stmt	Br	MC/DC
tcas	10	15	31	36	30	49	198	202	16	26	201	225
replace	30	46	163	43	82	130	349	380	93	148	388	411
tot_info	52	76	157	168	92	138	338	352	118	181	448	506
schedule	5	8	12	162	41	61	180	202	38	64	243	292
schedule2	0	0	0	34	14	20	48	58	6	9	115	121
print_tokens	6	9	56	0	20	29	160	168	24	39	231	252
print_tokens2	38	58	168	80	81	123	320	339	112	173	343	377
grep	208	306	615	783	363	595	842	926	469	722	900	955
sed	245	389	669	766	373	574	805	894	451	716	925	972
flex	208	306	666	822	362	517	824	981	427	712	894	912
gzip	194	324	698	764	302	503	834	892	461	709	944	988

Table 12
Standard deviations of the numbers of effective test suites for Ochiai.

Threshold value	$\alpha = 1\%$				$\alpha = 5\%$				$\alpha = 10\%$			
Adequacy criteria	Rand	Stmt	Br	MC/DC	Rand	Stmt	Br	MC/DC	Rand	Stmt	Br	MC/DC
tcas	3	0	6	9	5	6	10	8	2	2	5	4
replace	3	2	4	6	7	5	6	10	15	9	17	17
tot_info	2	3	5	4	8	4	6	10	13	8	5	6
schedule	0	0	0	0	3	0	4	5	6	4	3	9
schedule2	0	0	0	5	6	5	3	9	6	6	2	9
print_tokens	0	0	5	0	8	0	3	0	0	0	9	3
print_tokens2	14	8	18	6	5	7	6	2	5	3	9	9
grep	22	19	17	12	14	26	31	3	12	16	15	19
sed	17	17	21	19	16	29	5	14	15	9	19	23
flex	28	25	23	16	21	39	15	11	15	19	8	26
gzip	9	10	15	34	15	13	19	14	6	18	27	16

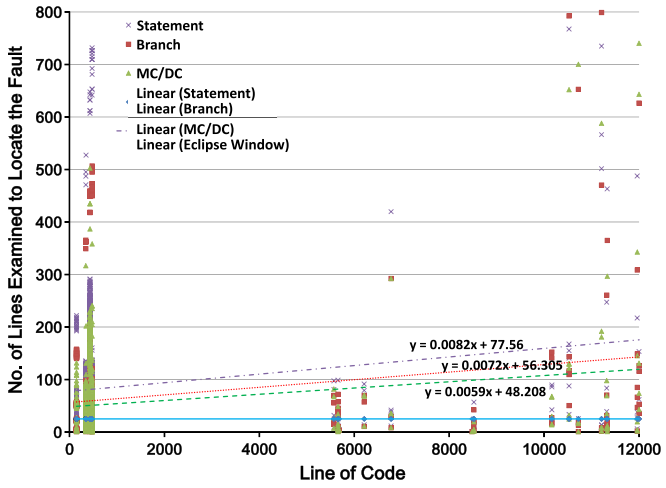


Fig. 1. Distribution of *Expense* with respect to size of faulty programs.

the two values ⟨LOC, No. of lines examined⟩ for a specific faulty version, and every regression line represents the impact of program size on the code examination effort for a specific test adequacy criterion.

Moreover, when using a fault localization tool as a debugging aid, developers would expect it to help them focus their attention on as small a suspicious code segment as possible; otherwise, the developers may lose patience and consider abandoning the use of the apparently ineffective tool. We observe that a typical debug window of an IDE (such as an Eclipse IDE or a Visual Studio IDE) may show around 25 lines of code without any scrolling. Using this number as a rough benchmark, we have drawn a solid (light blue[1]) horizontal line in Fig. 1.

Interestingly, we find from Fig. 1 that the most of the dots are distributed *above* the solid (light blue) line, and all the linear regression lines are also above that line. This indicates that in general, it is more likely that fault localization results based on the integration of the adequacy criteria and the statistical fault localization techniques under study do not help developers locate the fault within one debug window of a practical IDE.

Our analysis above represents the start of a new line of research, and the aim of this analysis is not to fully answer whether existing IDEs can effectively present information on statistical fault localization results to developers. However, the finding does raise interesting questions for future work. Can we design a test adequacy criterion that will likely construct test suites with effective fault localization results fitting into one screen? Alternatively, what kind of information should fit into a debug window to support effective fault localization? Also, what kinds of advances in human-computer interaction techniques (such as interactive presentation) will support effective fault localization of large applications?

The slopes of the regression lines are less than 0.0082, 0.0072, and 0.0059, respectively, indicating that the differences in marginal effects of statement-, branch-, and MC/DC-adequate test suites in supporting effective statistical fault localization are significant. Nonetheless, most data points are above the dotted line, indicating that a typical 25-line screen in an IDE may be ineffective in displaying the code that includes the faults.

Furthermore, we find that the slopes of the regression lines for statement-, branch-, and MC/DC-adequate test suites are around 0.0082, 0.0072, and 0.0059, respectively. For example, the slope for MC/DC is 0.0059, which implies that slightly less than 6 extra lines of code need to be examined for every 1000 LOC increase in program size. Moreover, the comparative slopes of the regression lines indicate that the differences in marginal effects of statement-, branch-, and MC/DC-adequate test suites in supporting effective statistical fault localization are significant. The linear equations for the regression lines can be approximated as $y = 0.0082 + 77$, $y = 0.0072x + 56$, and $y = 0.0059x + 48$. All the intercepts are positive numbers and more than 25, which indicate that there are overheads in locating faults even in small programs. Such overheads have typically exceeded the usual size of a 25-line screen in an IDE.

In the above discussion, we have also shown that adequate test suites outperform random test suites. Taking these two points into consideration, it appears that the use of the MC/DC adequacy criterion is the most promising choice in addressing the precision and scalability challenges in the integration issue, which we will further verify in the next section.

3.6.2. Answering RQ2

From RQ1, we know that using a MC/DC-adequate test suite seems better than using a branch-adequate test suite, which is in turn better than using a statement-adequate test suite. However, a stronger adequacy criterion (higher in the subsumption hierarchy) is usually associated with a larger test suite. Simply using more test cases alone can, on average, provide more information for fault localization. Thus, both the adequacy criterion and the test suite size may have influences on the effectiveness of statistical fault localization. We used a 2-dimensional plot of *Expense* with respect to test suite size for different adequacy criteria to see if one adequacy criterion is clearly a dominator of another, as shown in Section 3.6.2.1. To compare different test adequacy criteria more fairly, it is necessary to control the impact due to test suite size, which is further presented in Section 3.6.2.1. In addition, different adequacy criteria usually incur different test suite generation costs. Thus, we also measured the test suite generation cost for different adequacy criteria. We will discuss the cost-effectiveness of the studied adequacy criteria in Section 3.6.2.2.

3.6.2.1. Adequacy criteria vs. suite size. Because both test suite size and adequacy criterion are possible factors affecting the fault localization effectiveness of the test suites, we want to first analyze them by drawing 2-dimensional plots of *Expense* with respect to test suite size for the different adequacy criteria in Figs. 2 and 3, to see whether one adequacy criterion is clearly a dominator of another.

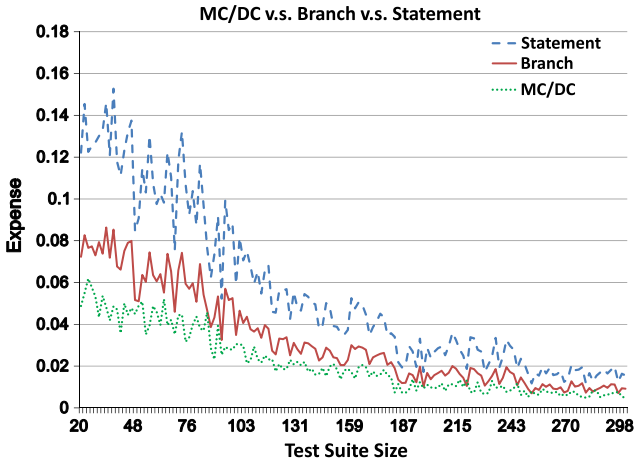
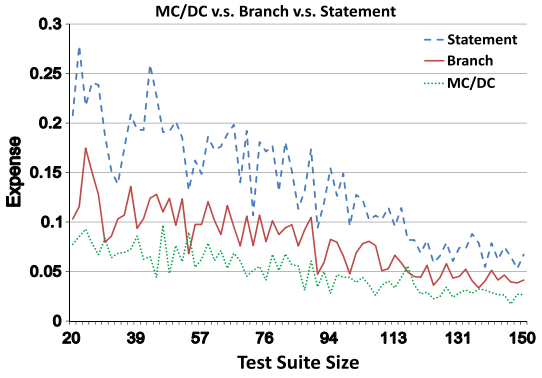
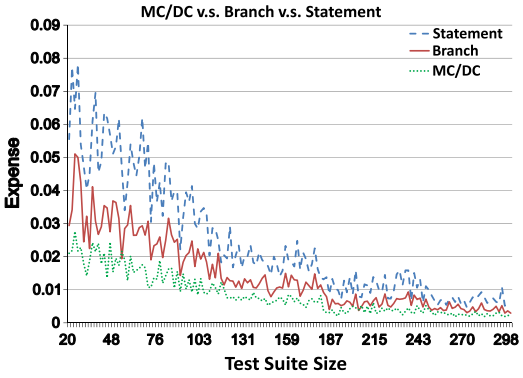


Fig. 2. Impacts of adequacy criteria and suite size on *Expense*.

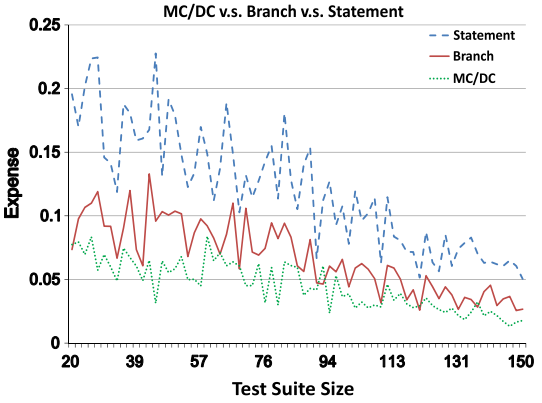
[1] For interpretation of color in Fig. 1, the reader is referred to the web version of this article.



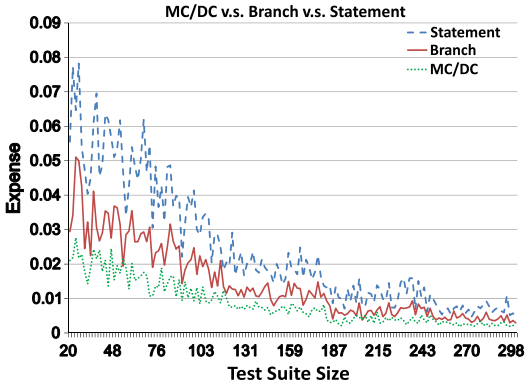
Siemens-Tarantula



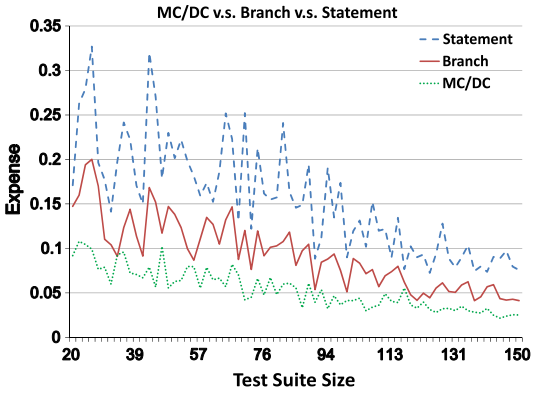
UNIX-Tarantula



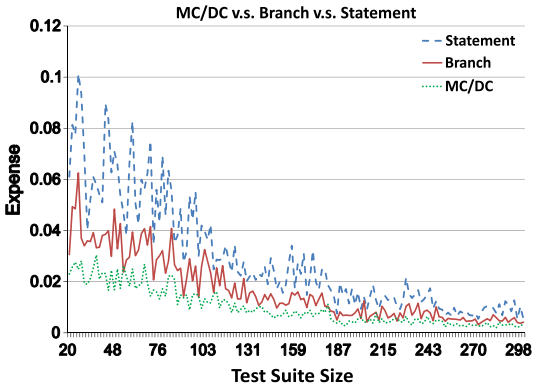
Siemens-SBI



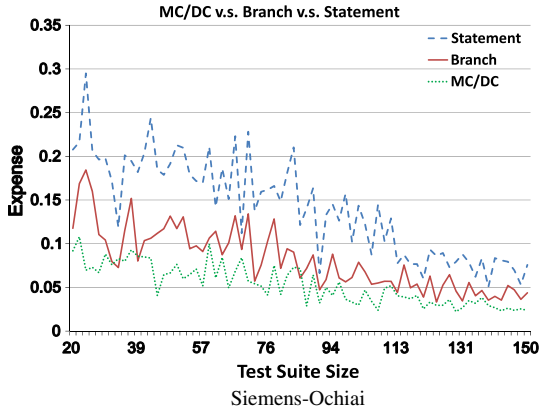
UNIX-SBI



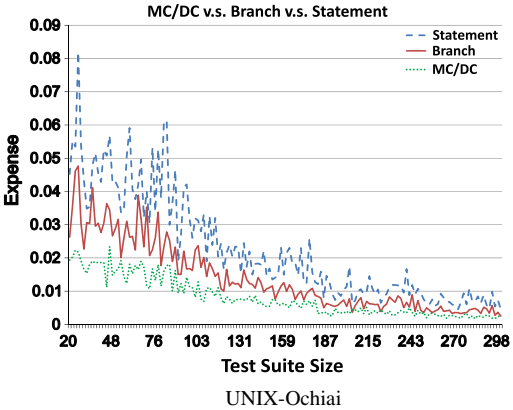
Siemens-Jaccard



UNIX-Jaccard



Siemens-Ochiai



UNIX-Ochiai

Fig. 3. Impacts of adequacy criteria and suite size on *Expense* for different fault localization techniques.

In Fig. 2, the *x*-axis is the test suite size, the *y*-axis is the *Expense* values of various test suites, and the three trend lines show the *Expense* values for statement-, branch-, and MC/DC-adequacy criteria with respect to test suite size over all programs, faulty versions, and fault localization techniques. Fig. 3 shows the same trend lines as those of Fig. 2, except that it shows the results for Siemens and UNIX programs on each fault localization technique separately.

If we fix the adequacy criterion, we can also see the trend that the *Expense* value tends to decrease gradually with larger test suite sizes. This finding affirms previous observations in the literature that larger test suite sizes *usually* incur better fault localization effectiveness. At the same time, Fig. 3 clearly shows that the trend is not smooth – along part of the curves, some smaller test suites can be more effective than larger ones in terms of *Expense*, with a difference of as much as 30 test cases in the test suite sizes.

We can see clearly from Figs. 2 and 3 that for a fixed test suite size, test suites satisfying the MC/DC-adequacy criterion have consistently lower *Expense* values than those satisfying the branch-adequacy criterion, which in turn have consistently lower *Expense* values than those satisfying the statement-adequacy criterion. If we fix the *Expense* value, we can see that MC/DC-adequate test suites, in fact, require fewer test cases than branch-adequate test suites, which in turn require fewer test cases than statement-adequate test suites. This clearly shows that statement adequacy, branch adequacy, and MC/DC adequacy are increasingly good at supporting effective fault localization.

3.6.2.2. Comparison of fault localization effectiveness. Tables 13–16 compare three pairs of strategies (random vs. statement, statement vs. branch, and branch vs. MC/DC) over all faulty versions for the Siemens and UNIX programs on Tarantula, SBI, Jaccard, and Ochiai, respectively. We summarize and compare what percentage of the adequate test suite for one criterion was better or worse than another in terms of *SavingRate*. The first row lists the pairs of adequacy criteria to be compared in the experiment. The second row shows the statistics to be compared for each group, namely, Rand for the random strategy, Stmt for statement adequacy, Br for branch adequacy, and MC/DC for MC/DC adequacy. The remaining rows show the values of the statistics (what percentage of one adequate test suite is better or worse than the other) across all the test suites for every program. For example, the row for tcas shows that among all the test suites for all the faulty versions of tcas, 89% of the statement-adequate test suites have better fault localization results than random test suites, 78% of the branch-adequate test suites have better fault localization results than statement-adequate test suites, and 80% of the MC/DC-adequate test suites have better fault localization results than branch-adequate test suites.

We first studied the comparison results for the three pairs of adequacy criteria in supporting statistical fault localization, considering random as a pseudo-criterion. From Tables 13–16, we observe that statement adequacy consistently performs better than the random strategy in terms of *SavingRate* across all programs and all fault localization techniques. Similarly, we also observe that MC/DC adequacy (or branch adequacy, respectively) consistently performs better than branch adequacy (or statement adequacy, respectively) in terms of *SavingRate* across all programs and all fault localization techniques.

Our observed results are consistent with our conjecture that among the adequacy criteria studied, the stronger the criterion (that is, subsuming others), the more effective it is for testing and debugging.

3.6.2.3. Discussion on cost-effectiveness. Different adequate test suites usually incur different test suite generation costs. In this section, we first measure the cost to select test cases from a test pool to generate an adequate test suite with respect to a test adequacy criterion, and then discuss the cost-effectiveness of different adequacy criteria. As discussed in previous sections, to generate statement-, branch-, and MC/DC-adequate test suites, we iteratively selected test cases from the test pool and added them into the test suite if the coverage achieved by the constructed test suite could be improved in terms of the criterion. The number of test cases within the adequate test suites used in our experiments varied from around 20 to 300 for different criteria and programs. The size of the random test suite ranged from $A/2$ to $A \times 2$, where A is the average size of the branch adequacy suites for the same program. When constructing the test suites in our experiment, we recorded and calculated the means and standard deviations of the test suite generation times (in ms) for different test adequacy criteria, as shown in Table 17.

We can see that the time taken to generate a random test suite is negligible. When comparing the mean test suite generation time for statement-, branch-, and MC/DC-adequate test suites, we can find that generating branch-adequate test suites incurs the smallest cost. Generating the MC/DC-adequate test suite incurs higher cost than generating statement-adequate test suite, which in turn involves higher cost than branch adequacy. We find the standard deviations of the time costs for generating different adequate test suites to be very close. Although different adequacy criteria involve different costs, the absolute time cost of the most expensive criterion (MC/DC) is not high for our subject programs. On average, our tool will take less than 23 s to generate a MC/DC-adequate test suites from the test pool. On the other hand, adopting stronger criteria like MC/DC will provide more precise debugging aid to save human debugging time, which can be very long. In general, trading affordable machine execution time for human code inspection time during debugging is quite worthwhile. This is because the former can run in the background while the latter is usually on the critical path in software development.

Table 13
Comparison of *SavingRate* for different adequacy criteria on Tarantula.

Subject	Random vs. statement (%)		Statement vs. branch (%)		Branch vs. MC/DC (%)	
	Rand ⩾ Stmt	Rand < Stmt	Stmt ⩾ Br	Stmt < Br	Br ⩾ MC/DC	Br < MC/DC
tcas	11	89	22	78	20	80
replace	9	91	10	90	35	65
tot_info	18	82	25	75	21	79
schedule	18	82	17	83	11	89
schedule2	15	85	22	78	23	77
print_tokens	9	91	20	80	24	76
print_tokens2	6	94	23	77	35	65
grep	17	83	20	80	32	68
sed	9	91	16	84	34	66
flex	6	94	14	86	16	84
gzip	5	95	16	84	31	69

Table 14
Comparison of *SavingRate* for different adequacy criteria on SBI.

Subject	Random vs. statement (%)		Statement vs. branch (%)		Branch vs. MC/DC (%)	
	Rand ⩾ Stmt	Rand < Stmt	Stmt ⩾ Br	Stmt < Br	Br ⩾ MC/DC	Br < MC/DC
tcas	19	81	24	76	34	66
replace	8	92	23	77	13	87
tot_info	7	93	10	90	21	79
schedule	14	86	23	77	31	69
schedule2	19	81	18	82	27	73
print_tokens	19	81	14	86	35	65
print_tokens2	17	83	17	83	20	80
grep	6	94	19	81	18	82
sed	7	93	12	88	22	78
flex	20	80	14	86	27	73
gzip	18	82	24	76	26	74

Table 15
Comparison of *SavingRate* for different adequacy criteria on Jaccard.

Subject	Random vs. statement (%)		Statement vs. branch (%)		Branch vs. MC/DC (%)	
	Rand ⩾ Stmt	Rand < Stmt	Stmt ⩾ Br	Stmt < Br	Br ⩾ MC/DC	Br < MC/DC
tcas	16	84	20	80	13	87
replace	17	83	24	76	32	68
tot_info	15	85	24	76	16	84
schedule	11	89	10	90	13	87
schedule2	9	91	17	83	35	65
print_tokens	19	81	23	77	14	86
print_tokens2	6	94	25	75	15	85
grep	18	82	21	79	14	86
sed	14	86	16	84	15	85
flex	20	80	22	78	21	79
gzip	16	84	14	86	11	89

Table 16
Comparison of *SavingRate* for different adequacy criteria on Ochiai.

Subject	Random vs. statement (%)		Statement vs. branch (%)		Branch vs. MC/DC (%)	
	Rand ⩾ Stmt	Rand < Stmt	Stmt ⩾ Br	Stmt < Br	Br ⩾ MC/DC	Br < MC/DC
tcas	6	94	25	75	29	71
replace	10	90	10	90	12	88
tot_info	13	87	25	75	30	70
schedule	14	86	14	86	33	67
schedule2	16	84	19	81	29	71
print_tokens	20	80	25	75	27	73
print_tokens2	12	88	24	76	28	72
grep	19	81	10	90	12	88
sed	11	89	19	81	28	72
flex	16	84	16	84	31	69
gzip	17	83	18	82	13	87

In general, a test case may be generated automatically, semi-automatically, or manually. Our result is applicable when comparing test suite construction costs under the automatic scenario. A further extension of the present study to cover the cost-effectiveness of testing-debugging integration for manual or semi-automatic test case constructions can be useful. To the best of our knowledge, the vast majority of existing research work on test adequacy criteria, test case prioritization, and statistical fault localizations do not deal with the human aspects of computing. They are certainly interesting to explore. One consideration for such explorations is that the experimentation requires controlling the content of the test pool among different test adequacy criteria to be compatible (if not identical) even when heterogeneous manual processes are involved. Furthermore, convincing show cases to demonstrate an effective integration between automatic test case generation and our work will also be crucial to make the research results more transferrable to the industry.

Table 17
Means and standard deviations of test suite generation times for different adequacy criteria.

Adequacy criteria	Random	Statement	Branch	MC/DC
Mean (ms)	0.01	12,988	8267	22,568
Standard deviation	<0.01	3550	2619	3815

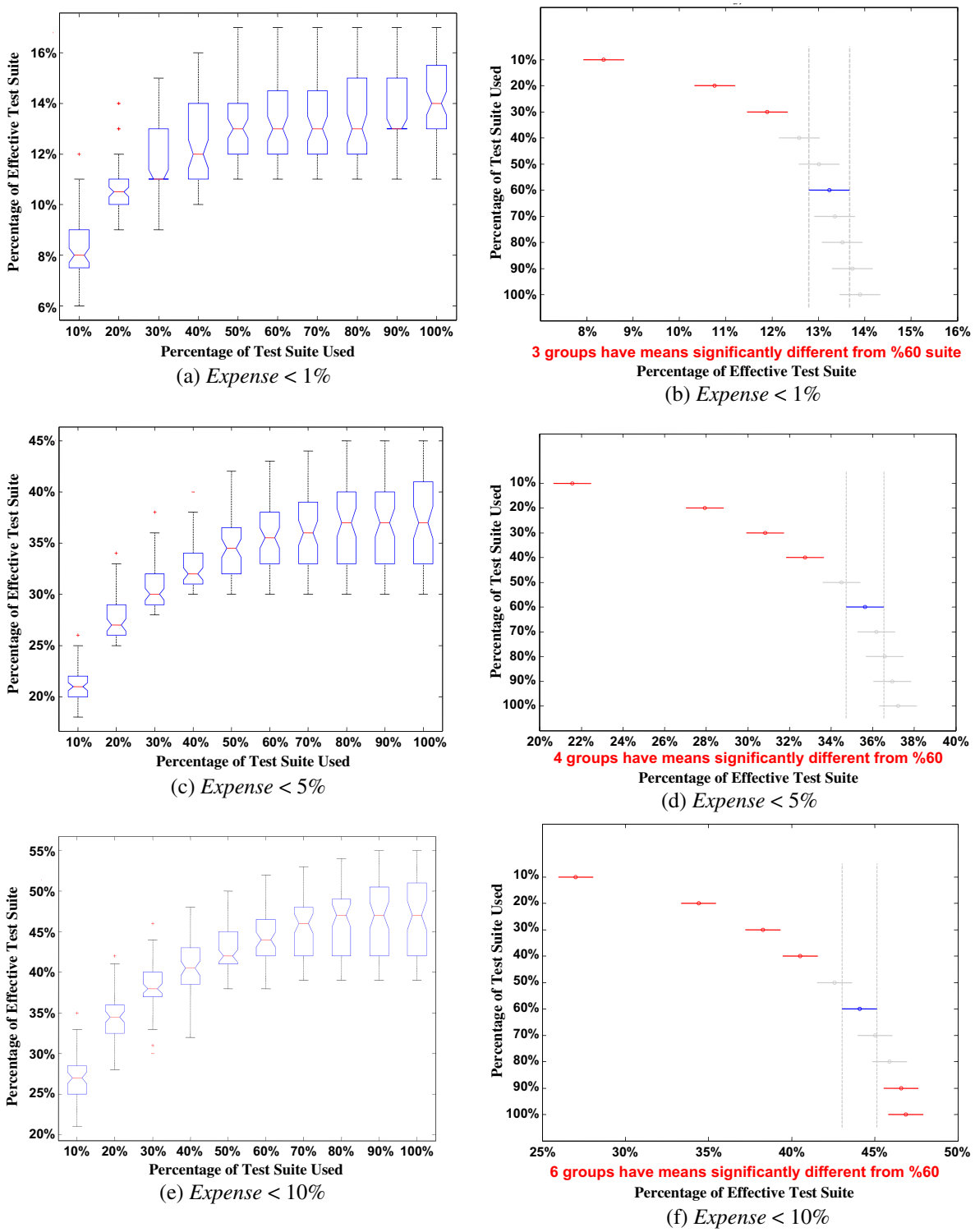
3.6.3. Answering RQ3

To answer RQ3, we conducted postmortem analysis on the integration results. Owing to the large number of possible standards to determine whether an integration is effective, we used three different threshold values of *Expense*, namely, $\alpha = 0.01$, 0.05, and 0.10, as the criteria to deem a test suite to be effective. They represent the cases that developers need to examine up to 1%, 5%, and 10% of the code in order to locate the faults. They were used in the conference version of this paper. In the present study, we analyze RQ3 based

on the two best adequacy criteria from RQ1 and RQ2: branch adequacy and MC/DC adequacy. When studying RQ3, we present the overall results of all prioritization techniques rather than showing each of them separately. We want to explore whether the adoption of test case prioritization techniques in general can be helpful for statistical fault localization.

3.6.3.1. Siemens programs with branch-adequate test suites. Figs. 4a, c, and e show the results of using branch-adequate test suites on Siemens programs. In each of these subfigures, the *x*-axis represents different percentages of a test suite used for fault localization while the *y*-axis represents the FLSP values (by examining the percentage of code up to the threshold value) for applying a test case prioritization technique before locating faults.

We observe from Fig. 4a that, by inspecting the top 1% of the ranked list of statements, the median *FLSP* value of a test suite is 8% if we prioritize and run the top 10% of a test suite for fault localization, which is very low. Even if we increase the percentage of



(a) *Expense* < 1%
(b) *Expense* < 1%
(c) *Expense* < 5%
(d) *Expense* < 5%
(e) *Expense* < 10%
(f) *Expense* < 10%

Fig. 4. The chance of test case prioritization techniques supporting effective fault localization using branch-adequate test suite for Siemens programs.

test suite to 100%, the median of the percentages of effective test suites is still less than 14%. The result indicates that it is still unlikely to locate the fault in the few (say, 1 to 5) top-ranked statements.

From Figs. 4a, c, and e, we observe that if a higher percentage of an original test suite is used for fault localization, the percentage of effective test suites increases. However, the increase is gradually less intense when the percentage of the test suite used reaches 60%. In particular, given a code inspection range of 1%, the use of 60% of the prioritized test cases for the fault localization already achieves a FLSP value of 13%, whereas the use of all the remaining 40% of test cases will only increase the *percentage* value up to 14%. We observe similar trends for code inspection ranges of 5% and 10% in Figs. 4c and e, respectively.

We also performed an ANOVA analysis to compare the mean FLSPs. The small p-value of 0.0032 consistently rejects the null hypothesis that the use of different percentages (namely, $10\%, 20\%, \ldots, 100\%$) of the same ordered test suites has the same FLSP values, at a significance level of 5%.

Figs. 4a, c, and e only show that there are differences in effectiveness when using various percentages of test suites for statistical fault localization, but they cannot tell whether they differ significantly. To see what percentage of test suites differ from one another in terms of FLSP, we further performed the multiple comparison procedure to find how different percentages of test suites differ significantly from one another at a significance level of 5%. Figs. 4b, d, and f show the results. The solid lines not intersected by the two vertical lines represent the percentages of test suites whose means differ significantly from the use of 60% of the suite for fault localization, while the gray lines represent the percentages of test suites comparable to the use of 60% of the suites for fault localization.

From Figs. 4b and d, we find that executing 60% of a test suite has no significant difference from executing the entire test suite. If we relax the code examination range to 10% for the Siemens suite, as shown in Fig. 4f, there will be a significant difference. It indicates that developers should estimate the amount of code they can afford to examine so that a test case prioritization technique can use it as a reference to determine the proportion of test suites to be executed.

3.6.3.2. UNIX utility programs with branch-adequate test suites. We also conducted the same postmortem analysis on the integration study for UNIX programs with branch-adequate test suites as we have presented in Section 3.6.3.1. Figs. 5a, c, and e show the results.

We observe from Fig. 5a that, by inspecting the top 1% of the ranked list of statements, the median FLSP value is 47% if we prioritize and execute the top 10% of a test suite for fault localization, which is much higher than that for the Siemens programs. Even if we increase the percentage of test suite to 100%, the median FLSP value is still under 65%. Although developers are willing to examine up to 5% (or 10%, respectively) of the code, Fig. 5c (or Fig. 5e, respectively) still shows that there is less than 65% (or 73%, respectively) of chance that the top 10% of test cases can assist them in locating faults effectively. The results show that developers should not greedily start fault localization based on a small percentage (10% in the above discussion) of the whole test suite.

The data show that there can be at least two strategies to address this problem. First, we observe across Figs. 5a, c, and e that since the corresponding bars among the three plots increase in terms of their y-values, it may be worthwhile to put in more effort in examining the code. Second, on each plot in Figs. 5a, c, and e, when a higher percentage of an original test suite is used for fault localization, the percentage of effective test suite increases remarkably. The results suggest that, if the preferred code examination range is fixed, the use of a higher percentage of test cases can be a good choice. It seems to us that this second strategy provides hints to answer the follow-up question in RQ1 that, in order to fit the code into one code-view screen, the use of a smaller adequate test suite for such testing-debugging integration may be a viable research direction. (However, the study on this aspect is outside the scope of this paper.)

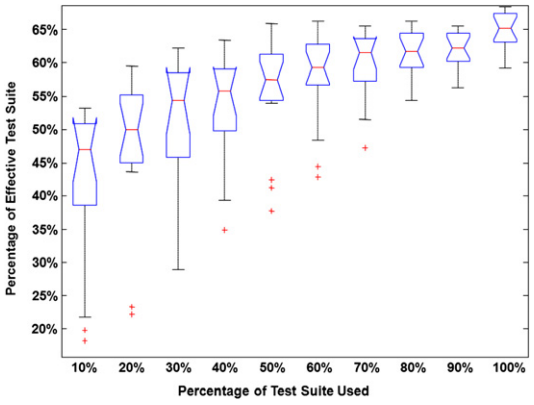
We perform ANOVA analysis to compare the mean FLSPs. The small p-value of 0.0142 rejects the null hypothesis (at a significance level of 5%) that the use of different percentages of test suites generates identical FLSP values. We further conduct the multiple comparisons procedure to find how different percentages of the same ordered test suites differ significantly from one another at a significance level of 5%. Figs. 5b, d, and f show the results. The solid lines not cut by the two vertical lines represent those percentages of test suites whose mean values differ significantly from the use of 100% of the suite for fault localization, while the gray lines represent those proportions of test suites whose effectiveness is comparable to the use of 100% of the suites for fault localization.

We observe from Fig. 5b that only when executing more than 60% of a test suite will there be no significant difference from executing the entire test suite in terms of FLSP. If we relax the code examination range to 5% and 10% of the code as shown in Figs. 5d and f, respectively, we still have the same results. It indicates that, for UNIX programs, around 60% of the test suite should be used to obtain fault localization effectiveness comparable to the use of the whole test suite. The results indicate that, if smaller test suites are used, the fault localization effectiveness is extremely likely to be decreased.

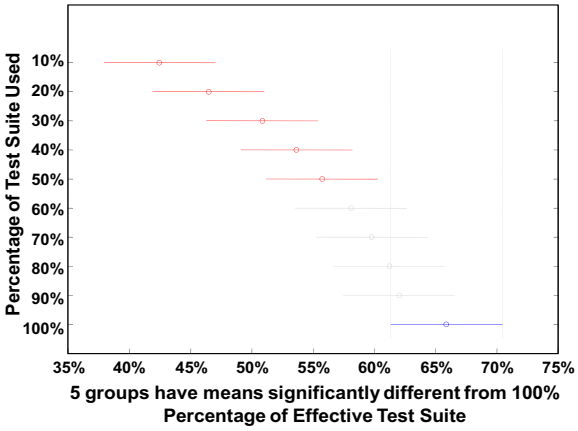
3.6.3.3. Siemens programs with MC/DC adequacy suites. Figs. 6a, c, and e show the corresponding results on the Siemens programs using the MC/DC-adequate test suites. The same procedure as described in Section 3.6.3.1 was used except that we used the MC/DC-adequate test suites rather than the branch-adequate test suites. We observe from Fig. 6a that by inspecting the top 1% of the ranked list of statements, the median FLSP value of a test suite is around 12.8% if we prioritize and execute the top 10% of the test suite for fault localization, which is still low. If we increase the percentage of test suite to 100%, the median percentage of effective test suites is still less than 22%. Similar to the result for branch-adequate test suites, the present result indicates that for the MC/DC-adequate test suites, it is also quite impractical to assume that the faults will be in the few (say, 1 to 5) top-ranked statements.

From Figs. 6a, c, and e, we find that if a higher percentage of an original test suite is used for fault localization, the percentage of effective test suites increases. However, the increase is gradually less noticeable when the percentage of the test suite used reaches 60%. In particular, given a code inspection range of <1%, the use of 60% of the prioritized test cases for the fault localization already achieves a FLSP value of 21%, whereas the use of all the remaining 40% of test cases will increase the *percentage* to 23% at most. We observe similar trends for code inspection ranges of <5% and <10% in Figs. 6c and e, respectively.

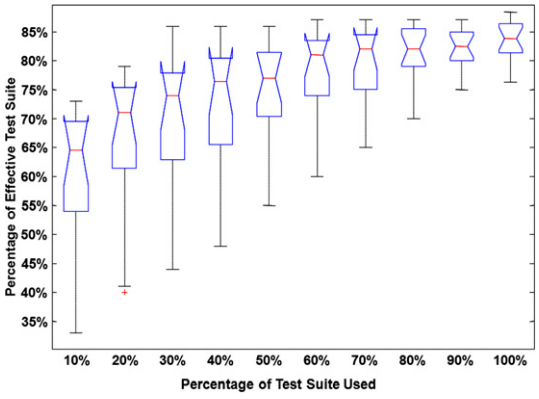
We conducted ANOVA analysis to compare the mean FLSPs. The p-value 0.021 rejects the null hypothesis that the use of different percentages (namely, $10\%, 20\%, \ldots, 100\%$) of the same ordered test suites has the same FLSP value, at a significance level of 5%. To see what percentages of test suites differ from one another in terms of FLSP, we further conducted multiple comparisons to find how different percentages of test suites differ significantly from one another, at a significance level of 5%. Figs. 6b, d, and f show the results. The solid horizontal lines not intersected by the two vertical lines represent the percentages of test suites whose mean values differ significantly from the use of 60% of the suite for fault localization, while the gray lines represent the percentages of test suites whose usage is comparable to the use of 60% of the suites for



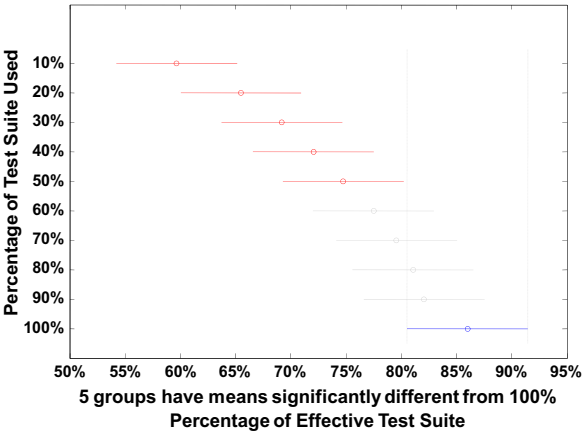
(a) *Expense* < 1%



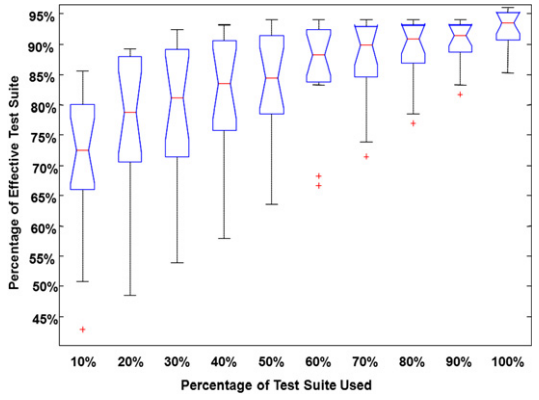
(b) *Expense* < 1%



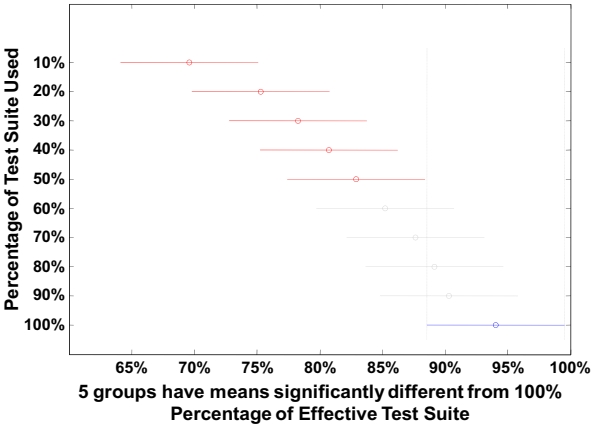
(c) *Expense* < 5%



(d) *Expense* < 5%



(e) *Expense* < 10%



(f) *Expense* < 10%

Fig. 5. The chance of test case prioritization techniques supporting effective fault localization using branch-adequate test suites for UNIX programs.

fault localization. We observe from Figs. 6b and d that executing 60% of a test suite has no significant difference from executing the entire test suite.

If we compare between Figs. 4 and 6, we can find that for both adequacy criteria, they show similar trends as a larger proportion of an adequate test suite is used for statistical fault localization. This implies that for the Siemens programs, around 40% of test suite can be avoided from execution without significantly compromising fault localization effectiveness.

There are also differences between MC/DC adequacy and branch adequacy in supporting fault localization. Let us compare the corresponding subfigures in Figs. 4 and 6. We find that larger proportions of MC/DC-adequate test suites are more effective in supporting fault localization than those of branch-adequate test suites. Our hypothesis testing confirms that the difference is statistically significant at a level of 5%. This is consistent with our earlier finding that MC/DC adequacy is stronger than branch adequacy in supporting effective fault localization.

3.6.3.4. UNIX utility programs with MC/DC adequacy suites. We also conducted postmortem analysis on the integration study for the UNIX programs using the MC/DC adequacy suites. Figs. 7a, c, and e

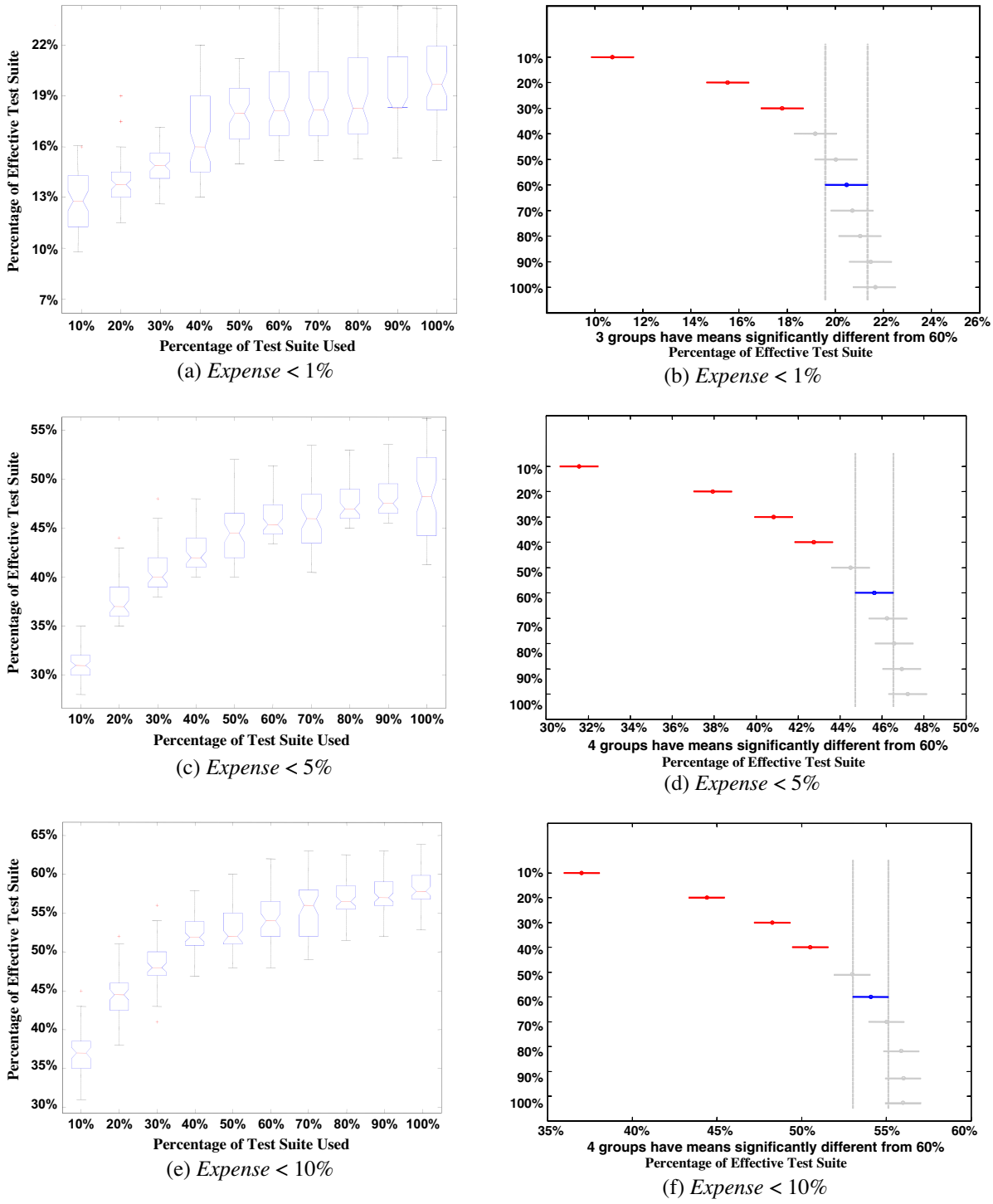
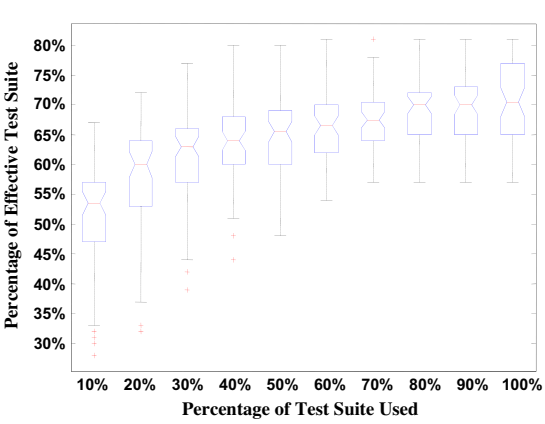


Fig. 6. The chance of test case prioritization techniques supporting effective fault localization using MC/DC-adequate test suite for Siemens programs.

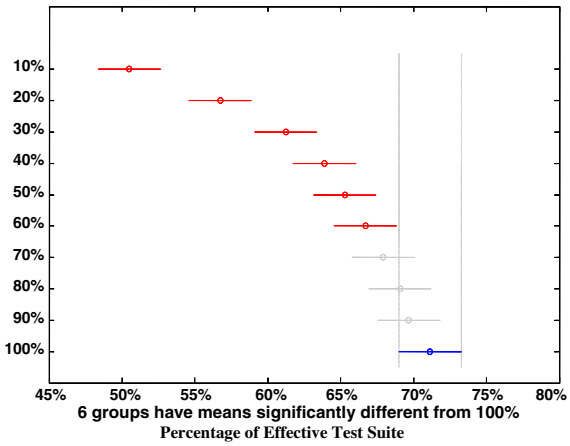
show the corresponding results. The meanings of the *x*- and *y*-axes are similar to the corresponding subfigures in Fig. 6.

We observe from Fig. 7a that by inspecting the top 1% of the ranked list of statements, the median FLSP value is 54% if we order a test suite by prioritizing the test cases and execute the top 10% of them for fault localization, which is much higher than that for the Siemens programs. Even if we increase the percentage of test suite to 100%, the median FLSP value is still under 75%. If we compare the results of Fig. 7 with those of Fig. 5, we find that fault localization effectiveness on the MC/DC-adequate test suites performs consistently better than that on the branch-adequate test suites. We also performed hypothesis testing to confirm that the difference is significant at a 5% significance level. This echoes our finding in earlier sections that stronger (subsuming) adequacy criteria can support statistical fault localization better than weaker (subsumed) ones.

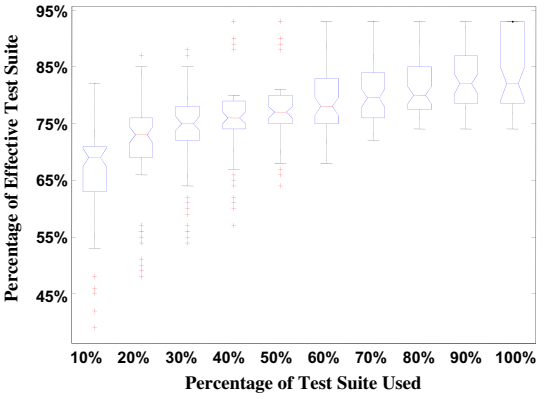
Similar to the discussions on Fig. 5, we observe from Fig. 7b that only when executing more than 70% of a test suite will there be no significant difference (in terms of FLSP) from executing the entire test suite. If we relax the code examination range to 5% and 10%



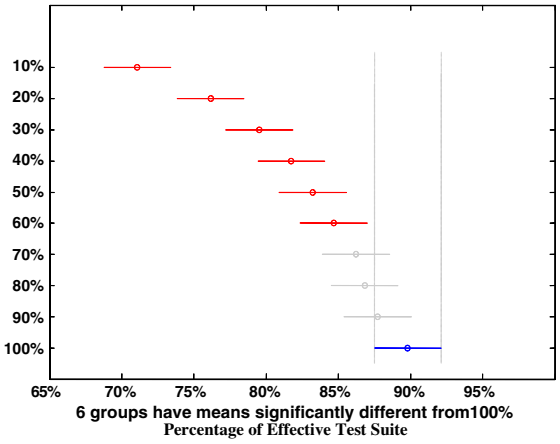
(a) *Expense* < 1%



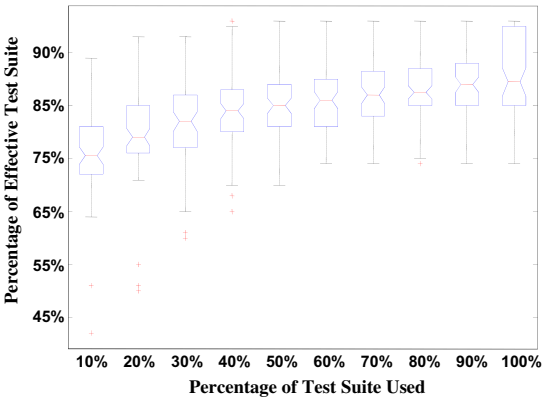
(b) *Expense* < 1%



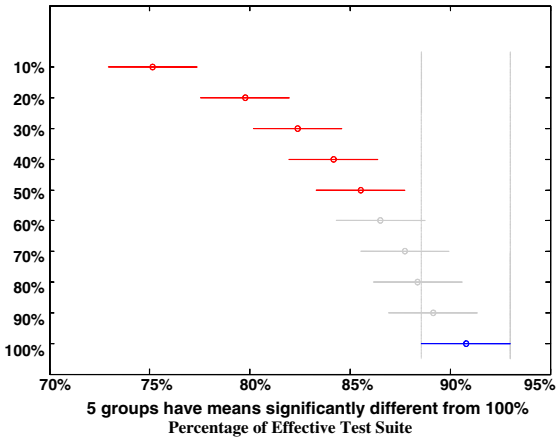
(c) *Expense* < 5%



(d) *Expense* < 5%



(e) *Expense* < 10%



(f) *Expense* < 10%

Fig. 7. The chance of test case prioritization techniques supporting effective fault localization using MC/DC-adequate test suite for UNIX programs.

of the code as shown in Figs. 7d and f, we still have similar results (70% and 60%, respectively). It shows that for the UNIX programs and the MC/DC-adequate test suites, around 60–70% of a test suite should be used to obtain a fault localization effectiveness comparable to the use of the whole test suite. The results indicate that using test case prioritization is highly recommended in the integration process as it can save as much as 40% of test case execution without affecting fault localization effectiveness.

We also looked into the test cases of the effective adequate test suites (including statement adequacy, branch adequacy, and MC/DC adequacy). We found that they are effective in locating faults due to several reasons. First, they may cover failure-revealing paths that are relatively difficult to cover in code coverage. Second, these adequate test suites on average have higher failure rates. Third, the branch or MC/DC coverage criterion has relatively more even coverage over all the possible paths, which makes the comparison between pass and fail test cases more precise and significant.

To conclude our finding, we can answer RQ3 that the probability of using a test case prioritization technique to generate effective

test suites for statistical fault localization is higher on the UNIX programs than on the Siemens programs. Furthermore, around 70% of a test suite should be used to retain the fault localization effectiveness of the whole test suite. Finally, we find that applying a stronger adequacy criterion not only indicates better testing effectiveness but also achieves better fault localization support.

3.7. Threats to validity

We used seven Siemens programs, four UNIX programs, and their accompanied faulty versions as our subjects. The use of other subject programs may result in different coverage patterns for failed test executions and passed test executions, which may result in different suspiciousness values assigned to the program statements. Although the set of faults cannot represent all possible faults, using them to conduct comparisons among techniques published in peer work is useful for researchers to compare results across different papers and experiments. Moreover, we used the adequate test suites provided by the SIR repository for Siemens programs and generated the branch- and statement-adequate test suites for the UNIX programs. We also generated MC/DC-adequate test suites for both Siemens and UNIX programs. The use of other adequate test suites may provide other results. We will leave the analysis and reporting of such test suites as future work.

In any case, our subjects have been widely used in existing test case prioritization, statistical fault localization, and regression testing research. Furthermore, branch- and MC/DC-adequate test suites have frequently been used in the experiments of testing and debugging papers. We believe that they have used these subjects in their experiments on solid basis with practical considerations. The results of our experiment complement their findings on these artifacts and help comparison across publications.

In this work, we used single fault versions of the subject program to perform the empirical study. On one hand, the single fault assumption is also frequently used in many other empirical studies. On the other hand, we recognize that a program may contain multiple faults in practice. Due to the tremendous scale of our current empirical study, we will leave the study of multi-fault versions as future work.

In our experiment, we excluded some faulty versions and test cases available from SIR. There are several reasons. The foremost reason is that in our experimental framework, we use gcov, a popular and freely available tool, to collect the branch and statement execution profile of each non-crashed execution. For crashed executions, gcov cannot provide coverage data. The techniques in our experiment, however, require coverage data in order to operate. Consequently, we excluded these test cases from the data analysis. As we have reported, our experimental environment was a UNIX server running Solaris. The C compiler on the underlying platform was provided by Oracle. Some versions could not be compiled. This was a platform-dependent issue and we removed these versions to reduce their impact.

Another reason for us to exclude some faulty version from the data analysis is that we followed previous papers on test case prioritization to conduct the experiment to exclude any version whose failures can be detected by more than 20% of the test cases in the test pool. The choice of this threshold value poses a threat to this study. Nonetheless, this practice has been widely used in the test case prioritization experiments. The use of this threshold value facilitates a comparison between this work and existing publications. A way to address this threat could be to conduct a larger experiment to vary this threshold from 0% to 100% systematically, and observe the effect. The effort to conduct this experiment and the corresponding data analysis are, however, overwhelming for us. We, therefore, excluded this aspect from our current experiment.

In Tables 5, 7, 9 and 11, the differences between the UNIX programs and the Siemens programs are dramatic. To avoid the internal validity caused by our subject programs, tools and results analysis procedures, we carefully checked and verified them, which confirmed the results. We believe that the program size, faults seeded, as well as other program features may explain these big differences, which we will explore as future work.

Another concern about the study may be the characteristics of the test suites. We used the test suites provided by SIR. They may not be representative in the sense that some test cases important to statistical fault localization may not be available. On the other hand, test case prioritization and fault localization are becoming mature and hence a common basis for comparison is necessary. To strike a balance between the use of more test suites and the comparability with a large body of published work, we chose the latter option in this study. In RQ1, we had 1000 branch-adequate test suites, 1000 statement-adequate test suites, 1000 MC/DC-adequate test suites, and 1000 random test suites for each subject program. They provided us with sufficient data points to compile statistical results shown in the paper. For RQ2, we would like to highlight that the results were based on one small test pool per subject program. As a result, we should not overly generalize the results. For some subject programs, the requirement of having branch- or MC/DC-adequate test suites may still be too demanding. For instance, almost all the subject programs used in the experiment reported in [5] did not come with test suites that are branch adequate or MC/DC adequate. We will leave this practical consideration as future work.

Another potential threat to validity is that the way we constructed adequate test suites may be different from that used in practice. It might be the case that a test suite constructed to target MC/DC has different features than one including test cases sampled from a pool until MC/DC is achieved. The evaluation of the impact of different adequate test suite construction strategy on fault localization effectiveness can be left as future work.

In this study, owing to time and resource limitation, we only evaluated random ordering, the coverage-based *Greedy*, and the white-box *ART*-based test case prioritization techniques. Although they are among the best general test case prioritization techniques studied in previous work, they have not been optimized. The use of optimized versions or other variants of these strategies as well as the use of other strategies may produce different results.

In drawing a comparison, we used the *Expense* metric, the *FLSP* metric, and the *SavingRate* metric. The use of other metrics may produce different results. The *Expense* metric has been widely used to evaluate statistical fault localization techniques. It, however, only represents one possible way of how developers may use the ranked list of statements, and it makes an assumption that any fault on each visited statement can be identified correctly with the same amount of effort. The time taken to evaluate such a statement and the precision of the fault identification has not been captured by this metric. The *FLSP* metric is built on top of the *Expense* metric. Owing to the limitation of the original metric, the effort to reveal a fault measured by the FLSP metric may not fully reflect the effort of developers to use the generated ranked list of statements to perform debugging. Readers are advised to interpolate the results of the experiment carefully. Finally, *SavingRate* is only one possible approach to discounting the influence of test suite size. Researchers may adopt other approaches to achieve the same goal.

Furthermore, the adequate test suite construction method for a stronger criterion such as MC/DC can be costly in practice. Our adequate test suite construction approach to selecting from a large test pool may incur less manual effort than that used in practice. Thus, there is a potential tradeoff: adopting a stronger adequacy criterion can improve fault localization precision and save debugging effort while it may also incur higher test suite construction effort. A

study of the issue using different adequate test suite construction approaches in the industry can further strengthen the validity of our empirical study, which is left as future work.

We also measured the means and standard deviations of different adequacy criteria. Our results show that different adequacy criteria have significantly different mean values in terms of SavingRate, although the absolute difference seems to be not large. The standard deviations are small. However, when the sizes of regression test suites are large (which is often the case in practice), the absolute difference will become much larger. To strengthen the validity of our results, we carefully verified our results to ensure the statistical difference was not due to large sample size.

4. Related work

Previous work has also studied the integration problem between testing and debugging.

Wong and colleagues proposed a technique to integrate test suite minimization and prioritization together [28]. Their heuristics is to select test cases based on the cost per additional coverage. Baudry et al. [4] used a bacteriologic approach to generating test cases in order to maximize the number of dynamic basic blocks. In this way, the fault localization techniques can be more effective. Yu and colleagues [29] explored the impact of using test suite reduction on fault localization. Their results show that test suite reduction does have an impact on fault localization effectiveness. However, test case prioritization differs from test suite reduction techniques in that test case prioritization is more flexible when allocating testing resources. If we use test case prioritization, the resources are used to execute the most important test cases, regardless of the time to stop. On the other hand, it is often the case that test suite execution must finish within a fixed time budget. As test suite reduction is criterion-based, it is difficult to fit into a changing testing budget. Another difference between our work and the above studies is that we focus on the differences among test adequacy and compare the effectiveness among such criteria. This dimension is new, and has not been studied in related work.

Jiang et al. [15] studied the integration problem of test case prioritization and fault localization. Their results show that test case prioritization does have an impact on the effectiveness of fault localization techniques and that the random strategy is surprisingly effective. However, the work did not study to what extent test case prioritizations may generate test suites that existing fault localization techniques can use to locate faults effectively. Neither did the work investigate the impact of test adequacy criteria on statistical fault localization. Gonzalez-Sanchez et al. [11] proposed a new test case prioritization approach to maximize the improvement of the diagnostic information per test case. Their results showed that their technique could reduce the overall testing and debugging cost for some scenarios. They also did not examine the effect of adequate test suites on fault localization techniques.

There are plenty of studies on test case prioritization techniques. Srivastava and Thiagarajan [26] proposed a binary matching technique to compute the changes between program versions at the basic block level and prioritize test cases to cover greedily the affected program changes. Li et al. [21] conducted evaluations of various search-based algorithms for test cases prioritization. Their results show that several search-based algorithms are surprisingly effective. Leon et al. [20] also proposed failure-pursuit sampling techniques. They are based on the observation that failure-inducing test cases tend to cluster together with respect to the code space of a program. Their failure-pursuit sampling uses one-per-cluster sampling to select the initial sample and, if a failure is found, its k nearest neighbors will be selected and checked. If additional failures are found, the process will be repeated.

There are also studies on fault localization techniques closely related to the four techniques studied in our experiment. For example, Cleve and Zeller [6] proposed delta debugging, which automatically isolates failure-inducing inputs, generates cause-effect chains, and exposes the faults. Renieris and Reiss [24] observed that using the execution trace difference between a failed run and its nearest passed neighbor run is more effective than using other pairs for fault localization. Jeffrey et al. [12] proposed a value-profile based approach to ranking program statements according to their likelihood of being faulty. Zhang et al. [32] differentiated short-circuit evaluations of individual predicates in individual program statements and produced a set of evaluation sequences per predicate for fault localization. They found that the use of evaluation sequence can significantly improve existing fault localization techniques. Zhang et al. [31] used a network propagation approach, taking into consideration the error propagation phenomena along the edges of a program control flow graph. They rank the edges of the program control flow graph and propagate back the suspicious scores to the program statements (representing states).

Since our study is an integration of test case prioritization techniques and fault localization techniques, the experiment will grow steeply when we evaluate more fault localization techniques. We therefore focus on the four most typical fault localization techniques in our study so that the empirical study is manageable without losing representativeness. Similarly, in RQ3, we narrow down our study to compare between MC/DC- and branch-adequate test suites. Although we have restrained the scale of our study, to the best of our knowledge, it is the largest empirical study on this topic to date.

Researchers have studied the MC/DC adequacy criterion for a long time because of its significance. Yu et al. [30] compared MC/DC, MUMCUT, and other related coverage criteria for safety-critical software by formal and empirical analysis. Since MC/DC is the required coverage criterion for airborne software by FAA through the DO-178B standard, it is also extensively studied in the aeronautics industry. Dupuy et al. [8] conducted an empirical evaluation of the MC/DC adequacy criterion on the HETE-2 satellite software. They found that test cases generated using the MC/DC-adequacy criterion detected important errors not detectable by functional testing. They further found that although MC/DC incurs more testing resources (40% of the total testing time), the effort is worthwhile as it can detect errors that could not have been found by lower level structural coverage. The relationship between test case prioritization and the MC/DC adequacy criterion has also been studied by Jones and Harrold [18]. Our work does not analyze this dimension whereas the work of Jones and Harrold does not study the fault localization aspect.

5. Concluding remarks

To select test cases from a huge input domain, testers use adequacy criteria to determine when to stop testing. Because the execution results and coverage information of the adequate test suite can be fed to fault localization techniques, the choice of adequacy criterion may have a significant impact on the effectiveness of fault localization.

We find from our study that stronger adequacy criteria may be more promising in supporting effective fault localization. In particular, we find that MC/DC adequacy performs better than branch adequacy, which in turn performs better than statement adequacy. Furthermore, we conducted postmortem analysis on existing fault localization techniques and found that they still could not effectively narrow down the suspicious region of faults within one debug window of typical IDEs. The result shows that there are still

large gaps in integrating various kinds of testing and debugging techniques so that they can be effectively used by developers uniformly. The result, however, indicates that MC/DC-adequate test suites can be more scalable and precise.

In terms of practice, there are a number of implications from the study. First, MC/DC is normally applied to safety-critical software. They have seldom been used in general. We have found from Tables 8–11 that as a stronger test adequacy is used, the probability of effective fault localization increases across all statistical fault localization techniques and across all subjects except applying Tarantula on the subject tot_info. In other words, the following conjecture holds, on average, for 97.7% of the 44 cases (four techniques with 11 subjects each):

- *Conjecture:* Using a stronger test adequacy increases the probability of effective fault localization via the use of a prioritized adequate test suite.

Based on this validated conjecture, we recommend the use of MC/DC instead of branch adequacy, statement adequacy, or random selection as the criterion to construct adequate test suite if the goal of the testing-debugging integration is effective statistical fault localization.

We have further discussed in Section 3.6.1 that using MC/DC test suites can make the regression line in Fig. 1 to have a gentler slope and a smaller y-intercept than using branch- and statement-adequate test suites – that is, it is more scalable and has less fixed cost. Nonetheless, to the best of our knowledge, only weaker criteria are popularly supported by many industrial-strength tools. While this may indicate the favor and demand of the software industry, we have conjectured that such popular adequacy criteria could be too weak for fault localization. In our experiment, the adoption of a stronger adequacy criterion can lead to more effective integration of testing and debugging.

On the other hand, random testing can be effectively used, say, to crash a program. We believe that random testing and adequacy testing are useful for different purposes and are complementary to each other. Our paper focuses on the impact of adequate test suites on fault localization. It will be interesting to study other popular and useful testing techniques (such as data flow testing) and resolve their effective integration. It will also be interesting to study reliability testing and its integration with program debugging.

Acknowledgements

This research is supported in part by the General Research Fund of the Research Grants Council of Hong Kong (Project Nos. 111410 and 717811), a Strategic Research Grant of City University of Hong Kong (Project No. 7002673), grants of the National Natural Science Foundation of China (Project Nos. 61003027 and 61202077), and the Fundamental Research Fund for Central Universities (Project No. YWF-12-LXGY-008). Part of this research was conducted when Tse was a visiting distinguished scholar at the State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China.

References

[1] Software considerations in airborne systems and equipment certification, DO-178B, RTCA, Washington, DC, 1992.
[2] The economic impacts of inadequate infrastructure for software testing, Final Report, National Institute of Standards and Technology, Gaithersburg, MD, 2002. <http://www.nist.gov/director/planning/upload/report02-3.pdf>.
[3] R. Abreu, P. Zoeteweij, A.J.C. van Gemund, On the accuracy of spectrum-based fault localization, in: Proceedings of the Testing: Academic and Industrial Conference: Practice And Research Techniques (TAICPART-MUTATION 2007), IEEE Computer Society, Los Alamitos, CA, 2007, pp. 89–98.
[4] B. Baudry, F. Fleurey, Y. Le Traon, Improving test suites for efficient fault localization, in: Proceedings of the 28th International Conference on Software Engineering (ICSE 2006), ACM, New York, NY, 2006, pp. 82–91.
[5] C. Cadar, D. Dunbar, D.R. Engler, KLEE: unassisted and automatic generation of high-coverage tests for complex systems programs, in: Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation (OSDI 2008), USENIX Association, Berkeley, CA, 2008, pp. 209–224.
[6] H. Cleve, A. Zeller, Locating causes of program failures, in: Proceedings of the 27th International Conference on Software Engineering (ICSE 2005), ACM, New York, NY, 2005, pp. 342–351.
[7] H. Do, S.G. Elbaum, G. Rothermel, Supporting controlled experimentation with testing techniques: an infrastructure and its potential impact, Empirical Software Engineering 10 (4) (2005) 405–435.
[8] A. Dupuy, N. Leveson, An empirical evaluation of the MC/DC coverage criterion on the HETE-2 satellite software, in: Proceedings of the 19th Digital Avionics Systems Conference (DASC 2000), vol. 1, IEEE Computer Society, Los Alamitos, CA, 2000, pp. 1B6/1–1B6/7.
[9] S.G. Elbaum, A.G. Malishevsky, G. Rothermel, Test case prioritization: a family of empirical studies, IEEE Transactions on Software Engineering 28 (2) (2002) 159–182.
[10] S.G. Elbaum, G. Rothermel, S. Kanduri, A.G. Malishevsky, Selecting a cost-effective test case prioritization technique, Software Quality Control 12 (3) (2004) 185–210.
[11] A. Gonzalez-Sanchez, E. Piel, H.-G. Gross, A.J.C. van Gemund, Prioritizing tests for software fault localization, in: Proceedings of the 10th International Conference on Quality Software (QSIC 2010), IEEE Computer Society, Los Alamitos, CA, 2010, pp. 42–51.
[12] D. Jeffrey, N. Gupta, R. Gupta, Fault localization using value replacement, in: Proceedings of the 2008 ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2008), ACM, New York, NY, 2008, pp. 167–178.
[13] B. Jiang, W.K. Chan, On the integration of test adequacy: test case prioritization and statistical fault localization, The 1st International Workshop on Program Debugging in China (IWPDC 2010), in: Proceedings of the 10th International Conference on Quality Software (QSIC 2010), IEEE Computer Society, Los Alamitos, CA, 2010, 377–384.
[14] B. Jiang, W.K. Chan, T.H. Tse, On practical adequate test suites for integrated test case prioritization and fault localization, in: Proceedings of the 11th International Conference on Quality Software (QSIC 2011), IEEE Computer Society, Los Alamitos, CA, 2011, pp. 21–30.
[15] B. Jiang, Z. Zhang, W.K. Chan, T.H. Tse, Adaptive random test case prioritization, in: Proceedings of the 24th IEEE/ACM International Conference on Automated Software Engineering (ASE 2009), IEEE Computer Society, Los Alamitos, CA, 2009, pp. 233–244.
[16] B. Jiang, Z. Zhang, W.K. Chan, T.H. Tse, T.Y. Chen, How well does test case prioritization integrate with statistical fault localization?, Information and Software Technology 54 (7) (2012) 739–758
[17] B. Jiang, Z. Zhang, T.H. Tse, T.Y. Chen, How well do test case prioritization techniques support statistical fault localization, in: Proceedings of the 33rd Annual International Computer Software and Applications Conference (COMPSAC 2009), vol. 1, IEEE Computer Society, Los Alamitos, CA, 2009, pp. 99–106.
[18] J.A. Jones, M.J. Harrold, Test-suite reduction and prioritization for modified condition/decision coverage, IEEE Transactions on Software Engineering 29 (3) (2003) 195–209.
[19] J.A. Jones, M.J. Harrold, J. Stasko, Visualization of test information to assist fault localization, in: Proceedings of the 24th International Conference on Software Engineering (ICSE 2002), ACM, New York, NY, 2002, pp. 467–477.
[20] D. Leon, W. Masri, A. Podgurski, An empirical evaluation of test case filtering techniques based on exercising complex information flows, in: Proceedings of the 27th International Conference on Software Engineering (ICSE 2005), ACM, New York, NY, 2005, pp. 412–421.
[21] Z. Li, M. Harman, R.M. Hierons, Search algorithms for regression test case prioritization, IEEE Transactions on Software Engineering 33 (4) (2007) 225–237.
[22] B. Liblit, M. Naik, A.X. Zheng, A. Aiken, and M.I. Jordan, Scalable statistical bug isolation, in: Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2005), ACM, New York, NY, 2005, pp. 15–26.
[23] C. Liu, X. Yan, L. Fei, J. Han, and S.P. Midkiff, SOBER: statistical model-based bug localization, in: Proceedings of the Joint 10th European Software Engineering Conference and 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering (ESEC 2005/FSE-13), ACM, New York, NY, 2005, pp. 286–295.
[24] M. Renieris, S.P. Reiss, Fault localization with nearest neighbor queries, in: Proceedings of the 18th IEEE International Conference on Automated Software Engineering (ASE 2003), IEEE Computer Society, Los Alamitos, CA, 2003, pp. 30–39.
[25] G. Rothermel, R.H. Untch, C. Chu, M.J. Harrold, Prioritizing test cases for regression testing, IEEE Transactions on Software Engineering 27 (10) (2001) 929–948.
[26] A. Srivastava, J. Thiagarajan, Effectively prioritizing tests in development environment, in: Proceedings of the 2002 ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2002), ACM, New York, NY, 2002, pp. 97–106.

[27] W.E. Wong, V. Debroy, B. Choi, A family of code coverage-based heuristics for effective fault localization, Journal of Systems and Software 83 (2) (2010) 188–208.
[28] W.E. Wong, J.R. Horgan, S. London, H. Agrawal, A study of effective regression testing in practice, in: Proceedings of the 8th International Symposium on Software Reliability Engineering (ISSRE 1997), IEEE Computer Society, Los Alamitos, CA, 1997, pp. 264–274.
[29] Y. Yu, J.A. Jones, M.J. Harrold, An empirical study of the effects of test-suite reduction on fault localization, in: Proceedings of the 30th International Conference on Software Engineering (ICSE 2008), ACM, New York, NY, 2008, pp. 201–210.
[30] Y.T. Yu, M.F. Lau, A comparison of MC/DC, MUMCUT and several other coverage criteria for logical decisions, Journal of Systems and Software 79 (5) (2006) 577–590.
[31] Z. Zhang, W.K. Chan, T.H. Tse, B. Jiang, and X. Wang, Capturing propagation of infected program states, in: Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT International Symposium on Foundations of Software Engineering (ESEC 2009/FSE-17), ACM, New York, NY, 2009, pp. 43–52.
[32] Z. Zhang, B. Jiang, W.K. Chan, T.H. Tse, X. Wang, Fault localization through evaluation sequences, Journal of Systems and Software 83 (2) (2010) 174–187.