Lightweight Fault-Localization Using Multiple Coverage Types

Raul Santelices,[†] James A. Jones,[‡] Yanbing Yu,[†] and Mary Jean Harrold[†]
[†]College of Computing, Georgia Institute of Technology
[‡]Department of Informatics, University of California, Irvine
[†]{raul|yyu|harrold}@cc.gatech.edu, [‡]jajones@ics.uci.edu

Abstract

Lightweight fault-localization techniques use program coverage to isolate the parts of the code that are most suspicious of being faulty. In this paper, we present the results of a study of three types of program coverage—statements, branches, and data dependencies—to compare their effectiveness in localizing faults. The study shows that no single coverage type performs best for all faults—different kinds of faults are best localized by different coverage types. Based on these results, we present a new coverage-based approach to fault localization that leverages the unique qualities of each coverage type by combining them. Because data dependencies are noticeably more expensive to monitor than branches, we also investigate the effects of replacing data-dependence coverage with an approximation inferred from branch coverage. Our empirical results show that (1) the cost of fault localization using combinations of coverage is less than using any individual coverage type and closer to the best case (without knowing in advance which kinds of faults are present), and (2) using inferred data-dependence coverage retains most of the benefits of combinations.

1. Introduction

Debugging is one of the most expensive and time-consuming processes for software developers. To address this expense, researchers have presented techniques to provide automated assistance in finding the faults that cause executions to produce incorrect outputs (i.e., *fail*). Many of these techniques monitor runtime events to find those events that correspond to the executions that fail. In particular, researchers have investigated using the runtime coverage of entities such as statements [1, 2, 8, 9] and branches [10, 11], which require lightweight instrumentation, and information flows [12], which require more expensive instrumentation. These researchers have shown empirically that techniques that use this coverage information provide guidance that can reduce the developer's effort in searching for the faulty parts of the software.

There are many types of coverage information that fault-localization techniques can utilize. However, to date there has been little research into which type of coverage maximizes fault-localization effectiveness or which type of lightweight coverage is best for fault localization in practical scenarios, such as deployed software [14], where monitoring overhead must be low. To understand the relationship among different types of lightweight coverage and the relative effectiveness of using them for fault localization, we performed, and present in this paper, an experiment on a set of Java programs. Our experiment explores the relative benefits of three lightweight types of runtime coverage monitoring—statements, branches, and du-pairs[1]—for use in the Tarantula [9] fault-localization technique. Tarantula runs a test suite on the target program, assigns suspiciousness scores to statements, and ranks the statements from most suspicious to least suspicious. To perform a quantitative comparison of all three coverage types, and to provide an understandable view of branch and du-pair suspiciousness, we created a method that measures the effectiveness of branches and du-pairs in terms of statements.

Our comparative experiment shows that the choice of coverage type can greatly affect the effectiveness of fault-localization: some faults are best localized by statements, others by branches, and still others by du-pairs. However, in general, it is impossible to know which kinds of faults exist in a program before finding these faults, and thus, which type of coverage is best (i.e., *ideal*) for fault localization. Therefore, given the results of this first study and our analysis of the contributions of each coverage type to fault localization, we developed a new technique that combines multiple types of coverage to leverage the strengths of the constituent types. This technique produces an aggregate fault-localization result in terms of statements.

To evaluate our combined scoring technique, we performed a second study on the same programs to determine whether fault-localization based on combined cover-

[1] A *du-pair* consists of a definition (i.e., assignment) d of a variable v and a use (i.e., read) u of variable v containing the value assigned at d.

age types is more effective than its constituent types and better approximates the ideal choice of coverage type per fault. This study showed that, for these subjects, combined coverage is more effective than statement, branch, and du-pair coverage applied individually. The study also showed that, across all faults studied, the effectiveness of using combined coverage is more *stable* (i.e., less variable) than using any individual coverage type.

An important concern in our work is runtime overhead. Studies [13, 16] show that statements and branches can be monitored efficiently (9%-18% overhead), but du-pairs have a much higher runtime overhead (66%-127%). To make fault localization as lightweight as possible, we applied a technique [16] that infers an approximation of du-pair coverage from branch coverage. To evaluate the fault-localization accuracy of inferred du-pair coverage, we performed a third study in which we compared full, more expensive du-pair coverage with our cheaper, approximate du-pair coverage. Our study shows that, for the same subjects, there is only a small loss in fault-localization effectiveness and stability when using the inferred du-pair coverage.

This work provides several benefits for fault-localization research. One benefit is that it shows that the type of coverage information for which the software is monitored does matter: different faults are found best by different coverage types. Another benefit is that it shows that, by combining different types of coverage information, both the overall cost and the variability of the fault-localization effort can be reduced. A third benefit is that du-pair inferencing lets runtime monitoring be efficient without greatly sacrificing effectiveness. A fourth benefit is that our combination can easily integrate into practice by utilizing only information from branch coverage, which is obtainable from common software-development and testing tools (e.g., *gcc*).

The main contributions of this paper are:

- A method for understanding and comparing the fault-localization effectiveness of different types of coverage in terms of statements and a study of fault-localization using this method for statements, branches, and du-pairs. The study shows that no coverage type is the most effective for all types of faults.
- A new combination technique that exploits the unique strengths of coverage types—statements, branches, and du-pairs. This technique can leverage du-pair coverage inferred from low-overhead monitoring with only a small loss of effectiveness for fault localization.
- A study demonstrating that, on average, combining coverage types improves the effectiveness of fault localization, better approximates the ideal choice of coverage per fault, and provides more stable fault localization than using individual coverage types.

2. Coverage-based Fault Localization

Researchers have proposed a number of fault-localization techniques based on coverage information provided by test suites. Such techniques (e.g., [1, 8, 9, 10]) typically instrument and execute the program with the test suite to gather runtime information. For each test case in the test suite, the instrumented program records the entities in the program (e.g., statements, branches, and du-pairs) that were executed (i.e., *covered*) and whether the test case passes or fails. This information is used to compute a heuristic measure for each program entity that expresses the *suspiciousness* of the entity as being responsible for test-case failures. These techniques typically sort the monitored entities in decreasing order of suspiciousness, which provides a *ranking* of statements from most to least suspicious. The results are then presented to the developer for guidance in finding the faulty code.

One such coverage-based fault-localization technique is Tarantula [9], which assigns to each statement a suspiciousness score between 0 and 1. Using the Tarantula technique, a number of formulas have been proposed for computing the suspiciousness of statements. Abreu and colleagues experimented with a number of different metrics, including the original Tarantula formula, and found that a similarity coefficient called *Ochiai*, often used in the molecular biology domain, was the most effective [1]. Our earlier experiments also showed that the Ochiai similarity metric was the most effective. Hence, in this work, we use the Ochiai formula computed for a statement *s*:

$$suspiciousness(s) = \frac{failed(s)}{\sqrt{totfailed * (failed(s) + passed(s))}}$$

where *totfailed* is the total number of failing test cases, *failed(s)* is the number of failing test cases covering *s*, and *passed(s)* is the number of passing test cases covering *s*.

To illustrate, consider program `mid()` and Fault 1 in Figure 1. For now, consider only the top group of rows labeled *Statements*. To the right of the code, for Fault 1, there is information about the six test cases: inputs are shown at the top of the columns, statement coverage is represented by bullets in the columns, and pass/fail status is shown at the bottom of the columns. Columns representing failing test cases are highlighted. To the right of the test-case columns, a column labeled *suspiciousness* shows the suspiciousness score calculated using the Ochiai formula. In this example, faulty statement 7 has the highest suspiciousness score of 0.71, giving that statement a ranking of 1.

To measure the *fault-localization cost* (i.e., the inverse of *effectiveness*) of a technique, we use the percentage of statements in the program that must be examined before reaching the first faulty statement; this measure has been used by other researchers (e.g. [5, 8, 15]) for fault-localization studies. For example, in Figure 1 and Fault 1, the ranking of statement 7 (which contains the fault) is 1, so the cost

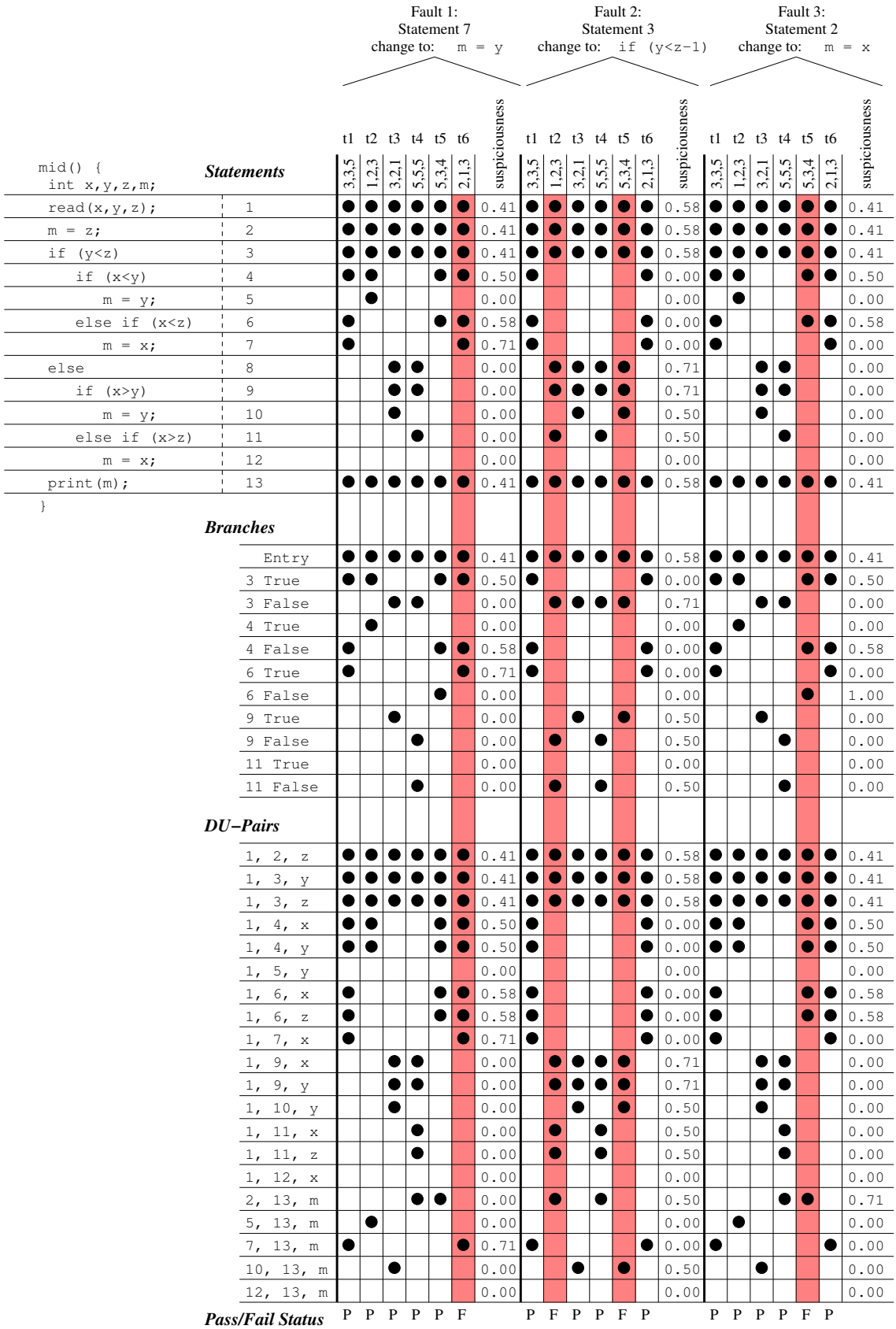

Figure 1. Example with three different faults with coverage and suspiciousness values for each.

Table 1. Suspiciousness scores for statements for all scoring approaches for `mid()` in Figure 1.

statement	**Fault 1** (statement 7)						**Fault 2** (statement 3)						**Fault 3** (statement 2)					
	st	br	du	max-SBD	avg-SBD	avg-BD	st	br	du	max-SBD	avg-SBD	avg-BD	st	br	du	max-SBD	avg-SBD	avg-BD
1	0.41	0.50	0.71	0.71	0.54	0.61	0.58	0.71	0.71	0.71	0.67	0.71	0.41	0.50	0.71	0.71	0.54	0.61
2	0.41	0.41	0.00	0.41	0.27	0.21	0.58	0.58	0.50	0.58	0.55	0.54	0.41	0.41	0.71	0.71	0.51	0.56
3	0.41	0.50	0.41	0.50	0.44	0.46	0.58	0.71	0.58	0.71	0.62	0.65	0.41	0.50	0.41	0.50	0.44	0.46
4	0.50	0.58	0.50	0.58	0.53	0.54	0.00	0.00	0.00	0.00	0.00	0.00	0.50	0.58	0.50	0.58	0.53	0.54
5	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
6	0.58	0.71	0.58	0.71	0.62	0.65	0.00	0.00	0.00	0.00	0.00	0.00	0.58	1.00	0.58	1.00	0.72	0.79
7	0.71	0.71	0.71	0.71	0.71	0.71	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
8	0.00	0.00	0.00	0.00	0.00	0.00	0.71	0.50	0.71	0.71	0.64	0.61	0.00	0.00	0.00	0.00	0.00	0.00
9	0.00	0.00	0.00	0.00	0.00	0.00	0.71	0.50	0.71	0.71	0.64	0.61	0.00	0.00	0.00	0.00	0.00	0.00
10	0.00	0.00	0.00	0.00	0.00	0.00	0.50	0.50	0.50	0.50	0.50	0.50	0.00	0.00	0.00	0.00	0.00	0.00
11	0.00	0.00	0.00	0.00	0.00	0.00	0.50	0.50	0.50	0.50	0.50	0.50	0.00	0.00	0.00	0.00	0.00	0.00
12	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
13	0.41	0.41	0.71	0.71	0.51	0.56	0.58	0.58	0.50	0.58	0.55	0.54	0.41	0.41	0.71	0.71	0.51	0.56
fault rank	1	2	3	4	1	1	6	2	4	4	4	2	6	6	3	4	5	4
loc. cost	7.7%	15.4%	23.1%	30.8%	7.7%	7.7%	46.2%	15.4%	30.8%	30.8%	30.8%	15.4%	46.2%	46.2%	23.1%	30.8%	38.5%	30.8%

of finding Fault 1 is 1/13, or 7.7%. When multiple statements share the same score, we use the approach reported in the literature: all tied statements get the greatest ranking-number for that set of statements. For example, for Fault 2 in Figure 1, statements 8 and 9, tied with score 0.71, have a ranking of 2; statements 1, 2, 3, and 13 (including faulty statement 3), tied with score 0.58, have a ranking of 6.

3. Evaluation of Individual Coverage Types

Our first objective was to evaluate and compare the fault-localization effectiveness of the coverage types we address in this paper: statements, branches, and du-pairs. In Section 3.1, we describe how we associate branch and du-pair coverage with statements, to permit comparison. In Section 3.2, we present our study of these three coverage types.

3.1. Mapping Coverage to Statements

To facilitate a comparison of the effectiveness of statements, branches, and du-pairs for fault localization, we developed a technique that maps branches and du-pairs to their related statements. This mapping lets us measure the fault-localization effectiveness of branches and du-pairs as percentages of statements that must be examined to find a fault. (Additionally, this mapping provides a statement-based suspiciousness view of the program which can be presented to the user with existing tools [9].)

Figure 2 illustrates the technique we use to compute fault-localization costs for each coverage type. For *statement coverage*, the technique first uses Tarantula to produce *statement scores* of suspiciousness, and then a Sorter produces a *statement ranking* using those scores. For *branch/du-pair coverage*, the technique first uses Tarantula to assign a suspiciousness score to each branch or du-pair (*branch/du-pair scores*). Then, the Mapper inputs these branch/du-pair scores and outputs *statement scores*. To produce the statement scores of suspiciousness, the Mapper first maps branches/du-pairs to statements using three rules.

Rule 1: Associate a branch (du-pair) with its conditional (definition) statement. A branch with a high suspiciousness score suggests that the condition in the branching statement might be incorrect, causing the incorrect branch to be taken at runtime. A du-pair with a high suspiciousness score suggests that the computation at the definition might be erroneous, causing the incorrect value to flow to the use and causing a failure.

Rule 2: Associate a branch (du-pair) with all statements that precede the conditional (definition) statement in the same basic block[2] and that perform intermediate computations that affect the condition (definition). Faults in such statements might produce erroneous intermediate values that make the affected branch (du-pair) cause a failure.

After applying Rules 1 and 2, some statements may still be left unmapped to branches (du-pairs). The Mapper uses Rule 3 to map those remaining unmapped statements.

Rule 3: For the branch mapping, associate an unmapped statement with all branches on which the statement is control dependent.[3] For the du-pair mapping, associate an unmapped statement with all du-pairs whose uses are located in that statement.

The Mapper then assigns to each statement the highest score of all branches (du-pairs) associated with that statement. The Mapper uses the highest score because a fault might cause the wrong associated branch (du-pair) to be executed; this branch (du-pair) produces failures and obtains a higher score than its alternative branches (du-pairs). The resulting *statement scores* are then input to the Sorter to create a *statement ranking* for branches (du-pairs). The Cost Calculator takes this ranking and the list of all *faulty statements* that constitute the fault, and outputs the *fault-localization cost* for that fault.

[2] A *basic block* is a sequence of statements with a single entry and exit.

[3] Statement A is *control dependent* on statement B if a branching decision at B determines whether A is necessarily executed or not.

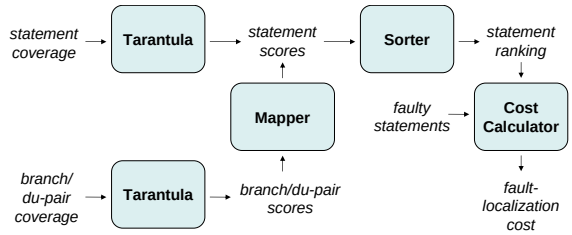

Figure 2. Computing fault-localization costs.

Table 1 shows, for the example in Figure 1, the statement scores for all three coverage types using our mapping. Columns *st*, *br*, and *du* correspond to statement, branch, and du-pair coverage, respectively; in Section 4, we discuss the other columns. The shaded rows show the scores for faulty statements. The last two rows show the ranking of the faulty statement (*fault rank*) and the cost of localizing the fault (*loc. cost*), respectively.

To illustrate our statement mapping and ranking based on branches, consider Fault 2 in Figure 1. The row group *Branches*[4] shows that Tarantula assigns scores of 0.00 and 0.71 to *3 True* and *3 False*, respectively. Using Rule 1, the Mapper associates these branches with faulty statement 3, and then assigns to statement 3 the higher of the scores for these two branches (i.e., 0.71). Using Rule 2, the Mapper also associates statement 1 with these branches because statement 3 is dependent on statement 1 and both are located in the same basic block. Thus, statement 1 is also assigned a score of 0.71. Because this score is the highest among all statements, statements 1 and 3 obtain a ranking of 2, resulting in a fault-localization cost of 15.4%. In contrast, for the same fault, the statement-coverage score for statement 3 is 0.58 with ranking 6, resulting in a fault-localization cost of 46.2%. Hence, branch coverage is more effective than statement coverage at localizing Fault 2.

To illustrate our statement mapping and ranking based on du-pairs, consider Fault 3 in Figure 1. The row group *DU-Pairs* shows that Tarantula assigns a score of 0.71 to *(2,13,m)*. The Mapper associates this du-pair with three statements: statement 2 using Rule 1, statement 1 using Rule 2, and statement 13 using Rule 3. Because 0.71 is the highest score associated with these three statements, the Mapper assigns all of them a score of 0.71 and a ranking of 3, resulting in a fault-localization cost of 23.1%. In contrast, for the same fault, the statement- and branch-coverage scores for statement 2 are both 0.41, ranking the fault 6 for both coverage types (with a fault-localization cost of 46.2%). For the faulty statement, the du-pair score is higher because test cases fail only when the wrong value assigned to m at statement 2 flows to statement 13 without being modified between those statements.

Finally, consider Fault 1 in Figure 1. The costs in the last row of Table 1 show that statement coverage localizes this fault better than branch and du-pair coverage, because only statement 7 gets the statement-coverage score of 0.71 (the highest), whereas more than one statement gets that same score from branch and du-pair coverage.

The results for Figure 1 illustrate that different faults can be best localized by different coverage types.

[4]Includes a special *Entry* branch representing entrance to the program.

Table 2. Subjects, test suite sizes, and faults.

subject	description	LOC	tests	faults
Tcas	collision avoidance	131	1608	10
Tot_info	information measure	283	1052	10
Schedule1	priority scheduler	290	2650	9
Schedule2	priority scheduler	317	2710	7
Print_tokens1	lexical analyzer	478	4130	5
Print_tokens2	lexical analyzer	410	4115	10
NanoXML v1	XML parser	3497	214	7
NanoXML v2		4009	214	7
NanoXML v3		4608	216	8
NanoXML v5		4782	216	7
XML-sec. v1	XML encryption	21613	92	7
XML-sec. v2		22318	94	7
XML-sec. v3		19895	84	2
JABA	program analyzer	37966	677	11

3.2. Study of Individual Coverage Types

The goal of this study is to compare statements, branches, and du-pairs for fault localization using our statement-mapping technique. First, we describe our empirical setup, and then we present and analyze the results.

3.2.1. Empirical setup

We used DUA-FORENSICS [16], which analyzes the Java bytecode language, instruments the program, and monitors the program's execution to collect coverage of statements, branches, and du-pairs. DUA-FORENSICS is based on the Soot analysis framework.[5] In addition, we implemented the process from Figure 2 to compute fault-localization costs.

Table 2 lists the subject programs, and for each program, provides a description, the number of non-blank and non-commented lines of Java code, the number of test cases, and the number of faults studied. These faults are unique and were seeded by other researchers. We excluded faults located in unreachable code or in code called exclusively from Java libraries (DUA-FORENSICS currently does not analyze libraries). We also excluded faults for which no available test case fails—at least one failing test case is necessary to reveal the presence of a fault. The first six subjects are part of the Siemens suite [7], which we translated from C to Java. For Tcas and Tot_info, we used only the first 10 faults to avoid biasing the average cost towards these two subjects. The remaining programs are used in real scenarios. We studied four releases of NanoXML and three releases of XML-security,[6] which we treat as separate sub-

[5]http://www.sable.mcgill.ca/soot.

[6]We obtained NanoXML and XML-sec. from SIR: http://sir.unl.edu.

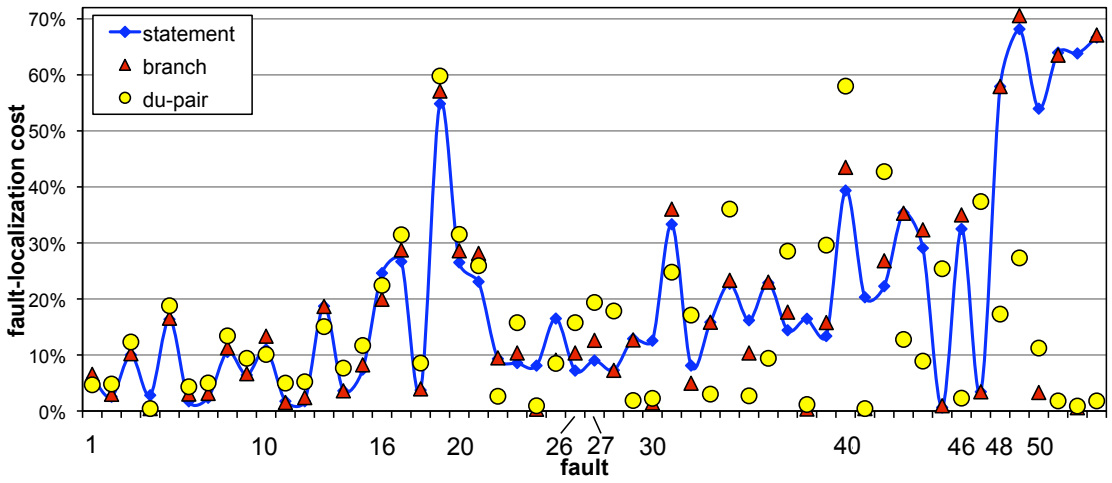

Figure 3. Faults with the greatest variations in fault-localization costs.

Table 3. Average fault-localization costs.

subject	statement	branch	du-pair	ideal
Tcas	22.41%	23.67%	14.86%	8.53%
Tot_info	24.99%	25.99%	20.09%	16.81%
Schedule1	5.78%	4.50%	11.23%	4.05%
Schedule2	32.30%	25.51%	19.40%	14.42%
Print_tokens1	14.48%	4.96%	10.19%	4.30%
Print_tokens2	5.40%	3.96%	4.54%	3.13%
NanoXML v1	3.25%	3.47%	4.54%	3.25%
NanoXML v2	4.64%	4.81%	3.85%	3.17%
NanoXML v3	6.10%	4.40%	3.05%	2.11%
NanoXML v5	5.06%	5.16%	3.83%	2.90%
XML-sec. v1	14.77%	11.44%	8.70%	8.10%
XML-sec. v2	8.44%	8.44%	9.62%	7.99%
XML-sec. v3	17.89%	17.86%	16.78%	16.78%
JABA	0.75%	0.82%	1.13%	0.70%
overall cost	11.49%	10.24%	9.02%	6.35%
standard dev	*16.25%*	*15.40%*	*12.04%*	*9.27%*

jects. In all, we studied 107 different faults.

For each fault, we instrumented the corresponding subject with DUA-FORENSICS, executed all test cases on the instrumented subject, collected the coverage of statements, branches, and du-pairs, and input this information to our Tarantula-based tool to produce the fault-localization costs.

3.2.2. Results and analysis

In this experiment, we examine the effectiveness in terms of fault-localization costs of each coverage type at three levels of granularity: overall (average for all 107 faults), per subject (2 to 11 faults per subject), and individual faults.

Table 3 shows the average fault-localization costs per subject (i.e., averaged over all faults in that subject) and overall (i.e., averaged over all faults in all subjects) when using statements, branches, and du-pairs (second, third, and fourth columns, respectively). The result for the best coverage type for each subject is shaded. The *ideal* cost for a single fault is the minimum of the three costs for that fault, corresponding to the best choice if we could correctly guess in advance which coverage type is most effective for that fault. Column *ideal* in Table 3 shows the average, per subject and overall, of the ideal costs. For example, for the seven faults in XML-sec.v1, the average fault-localization costs for statement, branch, and du-pair are 14.77%, 11.44%, and 8.70%, respectively; the average ideal cost is 8.10%.

Table 4. Summary of coverage types per fault.

coverage type	ideal		not ideal	
	# faults	avg. margin	# faults	avg. margin
statement	67	0.7%	40	13.8%
branch	22	0.8%	85	4.9%
du-pair	30	11.8%	77	3.7%

Table 4 shows a summary for each coverage type of the faults for which a coverage was the *ideal* choice and the faults for which it was *not ideal* (i.e., the localization cost was higher than the ideal). For column group *ideal* (*not ideal*), column *# faults* shows the number of faults for which a coverage was the ideal (not the ideal). Column *avg. margin* shows the cost margin, on average for all faults in the column group, by which the coverage was best (not the best). Specifically, for each fault, the margin for *ideal* is the difference in cost from the second-best coverage, and the margin for *not ideal* is the difference in cost from the ideal coverage. For example, statement coverage was the ideal choice for 67 faults by a cost margin of 0.7% on average and was not the ideal for 40 faults by 13.8% on average.

Figure 3 shows the fault-localization costs of using each coverage type for those faults in which the difference between the worst (highest) and best (lowest) cost is greater than 2%. The graph shows 53 faults, or about 50% of all faults. Faults are ordered by increasing difference between the worst and best costs, and numbered according to their position in that ordering. For readability, the costs for statement coverage are depicted by a solid curve. The costs for branch and du-pair coverage are shown as triangles and circles, respectively. This graph illustrates that different faults

are best localized by different coverage types. For example, faults 26 and 27 are best localized by statements, faults 16 and 50 are best localized by branches, and faults 46 and 48 are best localized by du-pairs.

Row *overall cost* in Table 3 shows that du-pairs are more effective at localizing these faults than branches, and branches are more effective than statements.[7] However, all three coverage types are far less effective, on average, than the ideal case. Du-pairs are the most effective type for eight subjects, branches are most effective on four subjects, and statements are most effective on two subjects. Table 4 shows that statements are, surprisingly, at least as good as the other types on 67 faults. In comparison, branches are at least as good as the other types on 22 faults, whereas du-pairs are at least as good as the alternatives on 30 faults. Although statement coverage performs better in more cases, it does so by a small margin—an average of 0.7% over the second-best type—and when statements are worse than the ideal, they are so by a large margin (13.8%). In contrast, du-pair coverage performs best in fewer cases, but for a much larger margin (11.8%), while performing worse than the ideal in the remaining cases by a smaller margin (3.7%).

The standard deviations of the costs for all faults, shown in row *standard dev* in Table 3, indicate that du-pairs have a noticeably lower variability in cost than statement and branch coverage, making du-pairs not only the most effective for fault localization, but also the most stable. However, the ideal case has an even lower variability than du-pairs. Hence, if we were able to predict the best coverage type for a given fault, we would obtain considerable gains both in cost and stability of the fault-localization effort.

Based on these results, for this set of subjects, test suites, and faults, we conclude that:

1. Our mapping of branches and du-pairs to statements is useful, letting these entities exhibit their expected benefits at fault localization with respect to statements.
2. Overall, du-pairs are more effective and stable at fault localization than branches, and branches are more effective and stable than statements.
3. Different faults are better found by different types of coverage. Without prior knowledge, no individual coverage type is the best choice for a fault or even for a group of faults in the same subject.
4. The ideal case is, overall, much better than any single coverage type. Therefore, there is ample room for reducing the cost of fault-localization.

4. Combination of Coverage Types

Motivated by the unique strengths of statements, branches, and du-pairs that we observed in our first study presented in Section 3.2, we created a new statement-scoring technique that leverages these strengths. The goal of our new scoring is twofold: reduce the overall cost of fault-localization, and mitigate the variability in cost (i.e., make the cost more predictable) for a set of faults. In Section 4.1, we present our combination approach. In Section 4.2, we present a study comparing this technique with individual types of coverage for fault localization.

4.1. Computing Combined Rankings

We present two main combination techniques: one based on the maximum of the scores for the individual coverage types and the other based on an average of the scores for the individual types of coverage.

4.1.1. Max-Statement-Branch-DU-Pair

In Section 3, for each individual coverage type, we assigned to each statement the score of the highest-scored entity of that type associated with the statement. Continuing with this strategy, we present our first combined scoring formula, *max-SBD*, which assigns to a statement the highest score among all entity types—statements, branches, and du-pairs—associated with that statement. For program `mid()` (Figure 1), the *max-SBD* columns in Table 1 present the scores for this combination. For example, for Fault 3, *max-SBD* gives a score of 0.71 to statements 1, 2 (the fault), and 13, which is only beaten by the score 1.0 of statement 6. As a result, the faulty statement gets a ranking of 4 and a fault-localization cost of 30.8%. This cost is worse than the cost of using du-pairs (23.1%) but better than using statements and branches (46.2% in both cases). For this fault, *max-SBD* provides a better result than randomly selecting one of the three individual types, which has an expected cost of 38.5%. However, *max-SBD* does not perform well with respect to individual coverage types for Faults 1 and 2. For all three faults, the average fault-localization costs when using statements, branches, du-pairs, and *max-SBD* are 33.3%, 25.6%, 25.6%, and 30.8%, respectively.

4.1.2. Average-Statement-Branch-DU-Pair

One problem with *max-SBD* is that, for each statement, it takes into account only the coverage type with the highest score, ignoring the contributions of the other two types, which might be better at localizing a particular fault. A high score assigned to a non-faulty statement through only one coverage type guarantees that such a statement is high in the ranking, thus hurting rather than helping the fault-localization effort. Thus, it is reasonable for each coverage type to have some weight in the combination scores. If all coverage types "agree" in assigning a high score to a statement, then there is supporting evidence for a high suspiciousness of that statement. Conversely, if some types give a statement lower scores than the other types, the suspiciousness of the statement should be adjusted to a lower

[7] Faults 29–53 in Figure 3 illustrate why du-pairs are best: for many faults in which the differences in fault-localization costs are the greatest, the cost of using du-pairs is lower than the alternatives.

Table 5. Difference from the ideal cost, per subject and overall, for individual types and combinations.

subject	statement	branch	du-pair	du-pair approx	max-SBD	max-SBD approx	avg-SBD	avg-SBD approx	avg-BD	avg-BD approx
Tcas	13.87%	15.14%	6.32%	15.07%	7.95%	16.74%	6.45%	13.35%	5.96%	13.53%
Tot_info	8.19%	9.18%	3.29%	2.58%	3.56%	2.85%	2.31%	1.31%	1.60%	0.08%
Schedule1	1.73%	0.45%	7.18%	6.30%	4.00%	2.49%	1.01%	0.97%	4.54%	3.81%
Schedule2	17.88%	11.09%	4.97%	6.51%	5.82%	6.80%	3.61%	5.77%	4.15%	6.41%
Print_tokens1	10.18%	0.66%	5.88%	17.42%	5.13%	6.15%	1.19%	5.02%	1.01%	3.96%
Print_tokens2	2.27%	0.83%	1.41%	4.21%	1.54%	4.04%	0.53%	1.54%	0.88%	2.08%
NanoXML v1	0.00%	0.21%	1.29%	2.60%	1.66%	2.76%	0.12%	0.43%	0.29%	0.74%
NanoXML v2	1.48%	1.64%	0.69%	1.07%	1.13%	1.16%	0.20%	0.28%	0.24%	0.58%
NanoXML v3	3.99%	2.29%	0.94%	0.88%	1.18%	0.96%	0.79%	0.45%	0.72%	0.44%
NanoXML v5	2.15%	2.25%	0.93%	1.42%	1.28%	1.66%	0.62%	0.92%	0.36%	0.75%
XML-sec. v1	6.67%	3.34%	0.61%	-1.11%	1.42%	-0.86%	1.65%	0.39%	1.47%	-0.15%
XML-sec. v2	0.45%	0.45%	1.63%	0.78%	0.97%	1.23%	0.10%	0.39%	1.23%	0.56%
XML-sec. v3	1.10%	1.08%	0.00%	1.16%	2.07%	2.11%	-0.28%	0.93%	-0.14%	1.07%
JABA	0.05%	0.12%	0.43%	0.48%	0.58%	0.62%	0.24%	0.25%	0.35%	0.36%
overall diff	5.14%	3.89%	2.68%	4.26%	2.77%	3.71%	1.48%	2.44%	1.80%	2.64%
standard dev	*13.28%*	*11.06%*	*5.50%*	*10.01%*	*5.03%*	*7.70%*	*4.03%*	*7.73%*	*4.66%*	*8.02%*

value, instead of relying on the maximum, as in *max-SBD*. Based on this intuition, we propose a second combined scoring formula, *avg-SBD*, which is the average of the scores of all three individual types. We also explore *avg-BD*, which is the average of the branch and du-pair scores only, motivated by the results of our initial study in which statements showed the lowest overall performance.

For the example in Figure 1, Table 1 shows in columns *avg-SBD* and *avg-BD* the scores of statements, the rankings of faulty statements, and the fault-localization costs using *avg-SBD* and *avg-BD*, respectively. For Fault 1, *max-SBD* assigns 0.71 to four statements, whereas *avg-SBD* and *avg-BD* assign 0.71 only to the fault and reduce the scores of the other three, non-faulty statements. These two combinations improve the ranking of the fault from 4 to 1 with respect to *max-SBD*. Overall, for all three faults in the example, *avg-SBD* performs as well as the best individual types (branches and du-pairs) with an average cost of 25.6%, whereas *avg-BD* performs even better, with an average cost of 17.9%.

4.2. Evaluating Combined Rankings

The goal of our second study was to evaluate the fault-localization effectiveness and variability of the combinations of coverage presented in Section 4.1.

4.2.1. Empirical setup

For this experiment, we used the same toolset described in Section 3.2.1 and the same subjects and faults listed in Table 2. We also added to our ranking tool the ability to score and rank statements based on combinations.

4.2.2. Results and analysis

Because our first study showed that individual types perform, overall, considerably worse than the ideal case, we wanted to investigate whether our combinations are closer to the ideal (i.e., the minimum cost of using statement, branch, and du-pair coverage), and, if so, how close they get. We can treat the ideal individual coverage type per fault as an approximation of the minimum cost that can be achieved with this kind of lightweight coverage information. Because different faults exhibit different levels of effectiveness for localization (see Table 3), the difference in cost from the ideal is a better measure of improvement than absolute costs. For these reasons, we focus in this study on the cost difference from the ideal individual type per fault. Nevertheless, to keep the absolute costs in perspective, we refer the reader to the costs for the ideal case in Table 3.

Table 5 shows the average difference in cost from the ideal, per subject and overall, for all fault-localization strategies. In this section, we consider six strategies: columns *statement*, *branch*, and *du-pair*, for individual coverage types, and columns *max-SBD*, *avg-SBD*, and *avg-BD* for our proposed combinations. Section 5 describes the remaining *approx* strategies. Table 5 shows that, overall, *max-SBD* performs slightly worse than one of its constituents, du-pair coverage, which incurs a cost difference from the ideal case of 2.68%. Also, for no subject was *max-SBD* more effective than the best individual coverage per subject. Therefore, *max-SBD* was not useful—it was better to use du-pairs. In contrast, using *avg-SBD* costs, overall, only 1.48% more than the ideal, which is closer to the ideal than any other strategy, including du-pair coverage. Also, on eight out of 14 subjects, *avg-SBD* performed better than its constituents. The third combination, *avg-BD*, also performed closer to the ideal than the individual types (1.80% overall), and better than its constituents on seven subjects. However, *avg-SBD* outperformed *avg-BD* overall and for eight subjects. The overall superiority of both *avg-SBD* and *avg-BD* over individual coverage types is statistically significant.[8] These results show that a combination of individual coverage types can indeed leverage the unique features of the constituents and achieve more effective fault localization, performing closer to the ideal of individual types without knowing in advance the best type for each fault.

For individual faults, Figure 4 presents the costs for the

[8] We used paired t-tests with p-values of 0.02 or less.

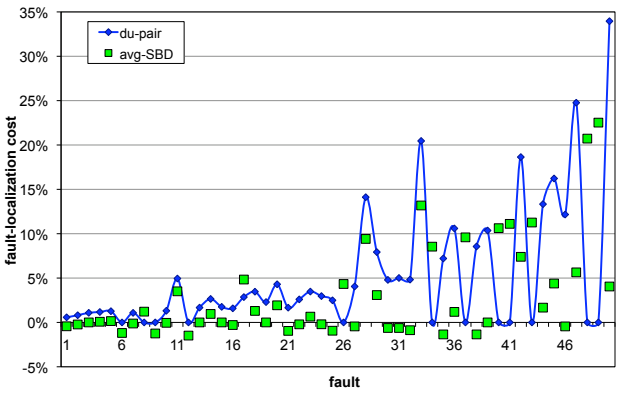

Figure 4. Comparison: *avg-SBD* vs. du-pairs.

best combination, *avg-SBD*, and the best individual coverage type, du-pairs. Because these costs are differences in cost from the ideal case for individual coverage, the horizontal line at 0% represents the cost of the ideal. (Whenever du-pairs are the ideal for a fault, the cost point for du-pairs is on the ideal line.) The graph shows results only for those 50 faults for which the difference between these two strategies is greater than 1%. Faults are ordered by increasing difference between the costs of these two strategies, and numbered by their position in that ordering.[9] The costs for du-pair coverage are depicted by a solid curve and the costs for *avg-SBD* are shown as squares. The graph shows that, more often than not, using *avg-SBD* is less expensive and closer to the ideal than using only du-pairs, which explains why *avg-SBD* is more effective overall. There are a few cases in which the combination is far less effective than du-pair coverage, such as faults 48 and 49 (where statement and branch coverage perform far worse than du-pair coverage, skewing the cost for *avg-SBD* up). However, these cases are more than compensated by cases in which *avg-SBD* performs far better, such as faults 44–47, and 50.

Interestingly, *avg-SBD* and *avg-BD* obtain a lower cost than the ideal for one subject—XML-security.v3—and several faults (see Figure 4). We observed that this phenomenon occurs for two reasons: (1) each constituent of the combination assigns a similar score to a faulty statement, and (2) many non-faulty statements that rank slightly higher than faulty statements for one constituent (adding to the cost when using only that constituent) get a low score from the other constituents, so in the combination such non-faulty statements end up ranking below faulty statements.

Based on these results, we can conclude that, for these subjects, test suites, and faults:

1. It is possible to leverage the fault-localization contributions of statement, branch, and du-pair coverage to create combinations, such as *avg-SBD* and *avg-BD*, that perform closer to the ideal individual case than any coverage type alone.
2. Although it is still not possible to prescribe a strategy that will be the most effective for individual faults, we can assert with statistical confidence that *avg-SBD* and *avg-BD* perform better than individual coverage, on average, for a sufficiently large set of faults.

[9] These fault numbers do not correspond to the numbering in Figure 3.

5. Approximation of DU-Pair Coverage

Our empirical studies of fault-localization (Sections 3.2 and 4.2) demonstrate the potential effectiveness of using du-pairs individually or with other coverage types. However, gathering du-pair coverage information can be expensive [13, 16]. In this section, we investigate the effects on fault localization of replacing du-pair coverage information with an approximation obtained from branch coverage. Branch coverage is considerably cheaper to obtain than du-pair coverage in terms of runtime overhead [13, 16], while being supported by existing tools. In Section 5.1 we provide background on du-pair coverage inferencing. In Section 5.2, we present our study on inferred du-pair coverage.

5.1. Inferring DU-Pair Coverage

Santelices and Harrold [16] presented a technique that infers an approximation of du-pair coverage from branch coverage information. The technique first infers which definitions and uses were covered at runtime and then, using additional static analyses, infers which du-pairs were definitely covered or not. Because the coverage of some du-pairs cannot be inferred with certainty from branch information alone for some executions, the technique reports such du-pairs in those cases as *possibly covered*.[10] In the next study, we simply treat possibly-covered du-pairs as covered.

5.2. Study of Inferred DU-Pair Coverage

The goal of this study is to determine whether strategies that replace du-pair coverage with du-pair information inferred only from branch coverage achieve better fault localization than statement coverage and branch coverage, which incur the same runtime overhead (i.e., branch monitoring).

5.2.1. Empirical setup

As in the previous two studies, we performed our experiment using the same toolset, subjects, and faults listed in Table 2. To infer du-pair coverage, we used the functionality already existing in DUA-FORENSICS [16].

5.2.2. Results and analysis

In this section, we present the results and analysis of the fault-localization costs and variability for the four strate-

[10] The study of this technique [16] revealed that of 85% of du-pairs reported as possibly-covered were actually covered.

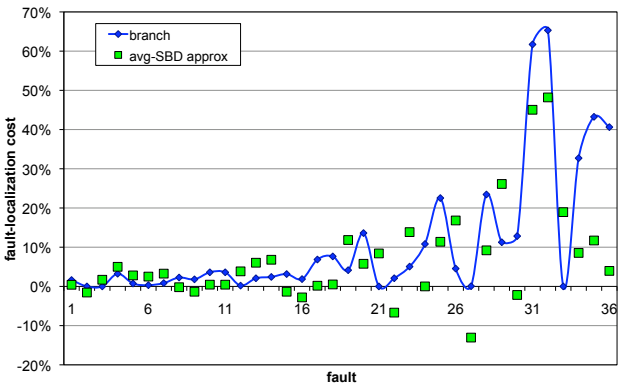

Figure 5. *avg-SBD approx* vs. branch coverage.

gies that require du-pair coverage—*du-pair*, *max-SBD*, *avg-SBD*, and *avg-BD*—after replacing du-pair coverage with the approximation inferred from branch coverage.

In Table 5, for each of the four columns involving du-pair coverage (i.e., *du-pair*, *max-SBD*, *avg-SBD*, and *avg-BD*), there is a corresponding column to the right with the qualifier *approx*, which shows the results for the same strategy using approximate du-pair coverage. For each of these strategies, the approximate version performs, overall and for most subjects, worse than the original version. For du-pairs, Table 5 shows that *du-pair approx* adds enough imprecision to make this strategy worse than branches, thus making *du-pair approx* not recommendable (i.e., it is better to use just branch coverage). The three approximate combinations, however, show an improvement over branches in both cost-proximity to the ideal and stability. Again, as in fault localization based on du-pair coverage (Section 4), *avg-SBD approx* is the best combination, whereas *max-SBD approx* is the worst. In particular, *avg-SBD approx* reduces the cost difference from the ideal when using branch coverage for fault localization from 3.89% to 2.44%, and reduces the standard deviation of this cost from 11.06% to 7.70%. The superiority in effectiveness of *avg-SBD approx* over branch coverage is statistically significant.[9]

Figure 5 illustrates the benefit of *avg-SBD approx* with respect to branch coverage for individual faults. The graph shows the differences in cost from the ideal individual choice for branch coverage as a solid curve and for *avg-SBD approx* as squares. The graph illustrates only those 36 faults for which the difference between the two strategies is greater than 1%. These faults are ordered by increasing difference in cost between the two strategies. The results show that, more often than not for these faults, *avg-SBD approx* is more effective than branches. The greater effectiveness of *avg-SBD approx* over branches is accentuated for those faults that exhibit the greatest difference in cost between the two strategies, such as faults 24–36. These results for individual faults help understand why *avg-SBD approx* performed better at fault-localization than branches, overall.

Interestingly, there are subjects for which the *approx* version of a strategy is more effective than the original version of the same strategy. For example, for Tot_info, *avg-SBD approx* is closer to the ideal than *avg-SBD*. This phenomenon might appear counter-intuitive, but it can be attributed to the heuristic nature of the fault-localization technique. Specifically, reporting a du-pair as possibly covered but not actually covered by failing test cases implies that at least the definition and use were covered, even if no value flowed between them. In such cases, the definition might obtain a suspiciousness score higher than for precise du-pair monitoring. If, by chance, the definition actually contains a fault, the effectiveness of the *approx* strategy can be improved. Overall, however, our results show that the imprecision introduced by inferred du-pair coverage, more often than not, increases the suspiciousness scores of statements unrelated to the fault with respect to the suspiciousness of the faulty statements.

Based on these results, we can conclude that, for these subjects, test suites, and faults:

1. Using approximate du-pair coverage inferred from branch coverage reduces, in general, the effectiveness of fault localization with respect to precise du-pair coverage. There are a few cases, however, in which this imprecision incidentally improves effectiveness.
2. Both *avg-SBD approx* and *avg-BD approx* are more effective and stable at fault localization than branch coverage, while incurring the same runtime overhead. Therefore, it is possible to extract more information from branch monitoring (e.g., approximate du-pair coverage) than just branch coverage in order to improve fault-localization over branch coverage alone.

6. Related Work

Many researchers have developed fault-localization techniques based on coverage or profiling information. In this section, we briefly survey this research.

Jones and colleagues [8, 9] developed the Tarantula technique that computes *suspiciousness* for each statement and ranks those statements. In recent work, Abreu and colleagues [1] compared the original Tarantula formula with the Jaccard and Ochiai coefficients. Their experiments showed that Ochiai, independent of test design, performs best for statement coverage. In this paper, we extended and studied Tarantula, using the Ochiai coefficient, for different entity types (i.e., branch and du-pair) and combinations of these types, for more effective and predictable (while still affordable) fault localization.

Masri [12] applied Tarantula to different coverage types, including branches and du-pairs, but, for most faults, his method could only utilize either branches or du-pairs, but not both. Therefore, a comparison of these two coverage

types was impossible. Nevertheless, his study showed that branches and du-pairs can be more effective at fault localization than statements. Masri also studied information flows, which were more effective at fault localization than branches and du-pairs. In this paper, we presented a way to map branches and du-pairs to all statements and faults, enabling a full comparison. Moreover, we combined these coverage types to provide more effective and predictable fault localization than individual coverage types. However, we did not consider information flows because they are much more expensive to monitor than lightweight entities, which are the focus of our work.

Dallmeier and colleagues [6] developed a lightweight fault-localization technique that uses sequences of method calls to localize faults at the class level. Liu and colleagues [11] developed a technique called SOBER that uses profiles (i.e., counts) of branch executions within each test case to localize branches that are related to faults. In contrast, our work localizes faults at the statement level and takes advantage of multiple, lightweight coverage types. Liblit and colleagues [10] developed SBI, which samples predicate coverage information and computes suspiciousness for certain types of predicates. However, they did not provide a way to combine those types.

More heavyweight approaches include a technique based on the Probabilistic Program Dependence Graph (PPDG) [3], by Baah and colleagues, which creates models by collecting data and evaluating predicates on executions. Retrieving and operating on program variables is, in our own experience on monitoring, considerably more expensive than collecting the coverage of du-pairs or branches.

7. Conclusion and Future Work

In this paper, we presented a method for comparing the fault-localization effectiveness of different lightweight coverage entities—statements, branches, and du-pairs. Using this method, we performed a study to empirically compare those entity types and concluded that different faults are better found by different types. On average, for all faults and also for most subjects, we presented two combinations that outperformed individual types. Although it is still not possible to prescribe a particular type or combination for individual faults, our study shows that, for a sufficiently large set of faults, our combination approach is the best choice because it comes closest to the ideal. Furthermore, we showed that using inferred du-pair coverage in combinations is more effective than branch coverage alone, while requiring the same runtime overhead, which is truly lightweight.

In the future, we will extend our experiments to include other relatively lightweight coverage types such as acyclic paths [4]. Also, because du-pairs can be more precisely inferred from acyclic paths, we expect to evaluate the gains in fault localization from using such paths.

Acknowledgements

This work was supported in part by NSF awards CCF-0429117, CCF-0541049, and CCF-0725202 and by a grant from Tata Consultancy Services, Ltd. to Georgia Tech.

References

[1] R. Abreu, P. Zoeteweij, and A. J. C. van Gemund. On the accuracy of spectrum-based fault localization. In *Proc. of TAIC-PART '07*, pp. 89–98, Sep. 2007.

[2] H. Agrawal, J. Horgan, S. London, and W. Wong. Fault localization using execution slices and dataflow tests. In *Proc. of Int'l Symp. on Softw. Reliability Eng.*, pp. 143–151, Oct. 1995.

[3] G. K. Baah, A. Podgursky, and M. J. Harrold. The probabilistic program dependence graph and its application to fault diagnosis. In *Proc. of Int'l Symp. on Softw. Testing and Analysis*, pp. 189–200, July 2008.

[4] T. Ball and J. R. Larus. Efficient path profiling. In *Proc. of 29th Int'l Symp. on Microarchitecture*, pp. 46–57, Dec. 1996.

[5] H. Cleve and A. Zeller. Locating causes of program failures. In *Proc. of Int'l Conf. on Softw. Eng.*, pp. 342–351, May 2005.

[6] V. Dallmeier, C. Lindig, and A. Zeller. Lightweight defect localization for java. In *Proc. of European Conf. on Object-Oriented Programming*, pp. 528–550, July 2005.

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

[8] J. A. Jones and M. J. Harrold. Empirical evaluation of the Tarantula automatic fault-localization technique. In *Proc. of Int'l Conf. on Automated Softw. Eng.*, pp. 273–282, Nov. 2005.

[9] J. A. Jones, M. J. Harrold, and J. Stasko. Visualization of test information to assist fault localization. In *Proc. of Int'l Conf. on Softw. Eng.*, pp. 467–477, May 2002.

[10] B. Liblit, M. Naik, A. X. Zheng, A. Aiken, and M. I. Jordan. Scalable statistical bug isolation. In *Proc. of Conf. on Progr. Lang. Design and Impl.*, June 2005.

[11] C. Liu, X. Yan, L. Fei, J. Han, and S. P. Midkiff. SOBER: statistical model-based bug localization. In *Proc. of European Softw. Eng. Conf. and Foundations of Softw. Eng.*, pp. 286–295, Sep. 2005.

[12] W. Masri. Fault localization based on information flow coverage. *Technical report: AUB-CMPS-07-06. American University of Beirut, Computer Science*, Aug. 2007.

[13] J. Misurda, J. A. Clause, J. L. Reed, B. R. Childers, and M. L. Soffa. Demand-driven structural testing with dynamic instrumentation. In *Proc. of Int'l Conf. on Softw. Eng.*, pp. 156–165, May 2005.

[14] A. Orso, D. Liang, M. J. Harrold, and R. Lipton. Gamma system: Continuous evolution of software after deployment. In *Proc. of Int'l Symp. on Softw. Testing and Analysis*, pp. 65–69, July 2002.

[15] E. Renieris and S. Reiss. Fault localization with nearest neighbor queries. In *Proc. of Int'l Conf. on Automated Softw. Eng.*, pp. 30–39, Oct. 2003.

[16] R. Santelices and M. J. Harrold. Efficiently monitoring data-flow test coverage. In *Proc. of Int'l Conf. on Automated Softw. Eng.*, pp. 343–352, Nov. 2007.