A Study on the Effect of Coincidental Correctness on Fault Localization Metrics

Rawad Abou Assi, Wes Masri, and Chadi Trad
Electrical and Computer Engineering Dept.
American University of Beirut
{ria21, wm13, cht02}@aub.edu.lb

***Abstract*— According to the PIE model, three conditions must be met for failure to be observed: 1) the defect is executed, 2) the program is infected, and 3) the infection has propagated to the output. Weak coincidental correctness (CC) occurs when the program produces the correct output, while condition 1) is satisfied but 2) and 3) are not satisfied. Strong coincidental correctness occurs when a correct output is observed, while both conditions 1) and 2) are satisfied but not 3).**

In prior work, we analytically demonstrated that CC is a safety reducing factor for coverage-based fault localization (CBFL). However, we did not experimentally validate that fact, which we do in this paper. Specifically, we comparatively evaluated the performance of CBFL using ten different suspiciousness metrics when: a) both weak and strong CC tests are present; b) no weak nor strong CC tests are present; c) only weak CC tests are present; d) only strong CC tests are present. Our experiments showed that when the CC tests are discarded, in most cases the suspiciousness score of the defective statement increased and its EXAM ranking score also improved. The metrics that benefited most from discarding CC tests are: *Tarantula, Ample, Ochiai, Dstar*^{2}, and *Dstar*^{3}**. Whereas, discarding CC tests had no effect on *Russel*, *Wong1*, and *Binary*. However, the latter three metrics were the worst performers in regard to the EXAM score.**

Keywords— coverage-based fault localization, coincidental correctness, suspiciousness metrics, failed error propagation, fault masking

I. INTRODUCTION

Most coverage-based fault localization (CBFL) techniques assume that the execution of a defective code location will lead to a program failure [1][2][12][13][14][15][16][26]. However, this may not always be the case.

According to the PIE model [27], three conditions must be met for failure to be observed: 1) the defect is **e**xecuted, 2) the program is **i**nfected, and 3) the infection has **p**ropagated to the output. Amman and Offutt supported this same notion in their RIP (reachability-infection-propagation) model described in [3]. *Coincidental correctness* (CC) [8][10][28] occurs when the program produces the correct output, while conditions 1) and 2) are satisfied but not 3). We refer to this case as *strong coincidental correctness* to differentiate it from *weak coincidental correctness* which occurs when the program produces the correct output, while condition 1) is satisfied but 2) and 3) are not satisfied [17][18][19].

Several researchers have recognized the negative impact of coincidental correctness on the effectiveness of defect detection techniques [4][5][6][7][9][10][28]. In previous work, we showed [17][18][19] that both weak and strong CC are prevalent. We also analytically demonstrated that weak CC is a safety reducing factor for CBFL; i.e., when weak CC tests are present, the defect will be assigned a suspiciousness score smaller than when they are not present. In this paper, we experimentally assess the impact of coincidental correctness on the effectiveness of CBFL using ten different suspiciousness metrics. We do so by considering test suites that fall under the following four categories:

1) $\boldsymbol{T}_{CC_{w+s}}$: both weak and strong CC tests are present.
2) $\boldsymbol{T}_{CC_{\emptyset}}$: no weak nor strong CC tests are present.
3) $\boldsymbol{T}_{CC_{w}}$: only weak CC tests are present.
4) $\boldsymbol{T}_{CC_{s}}$: only strong CC tests are present.

Since our experiments involve ten different metrics and four test suite categories, this paper sheds light on 40 different potential environmental setups.

Section II describes the used CBFL approach and metrics. Section III presents our experimental results. Section IV briefly surveys related work, and Section V concludes.

II. COVERAGE-BASED FAULT LOCALIZATION

This section describes the CBFL techniques evaluated in our experiments, namely, the profile elements, the suspiciousness metrics, and the ranking approach.

A. Profiles

The execution profiles we used encompass *def-use* pair (DUP) coverage information. Specifically, for every pair consisting of a variable *definition D(x)* and a *use U(x)* such that *D(x)* dynamically reaches *U(x)* in at least one test, a DUP profile contains a flag indicating whether *D(x)* dynamically reaches *U(x)* in the current test. The suspiciousness score of a given DUP is assigned to its component statements, i.e., to both the *def* and *use* statements. The choice of *def-use* pair coverage is completely arbitrary, as we might have used any other type of coverage, such as statement or branch.

B. Suspiciousness metrics

This work considers the following ten CBFL suspiciousness metrics that were used and/or studied in the literature: 1) *Tarantula* [12][13]; 2) *AMPLE* [7]; 3) *Ochiai* [2]; 4) *Dstar*^{2}; 5) *Dstar*^{3}; 6) *Naish1*; 7) *Naish2*; 8) *Wong1*; 9) *Russel*; and 10) *Binary*. The first three metrics are among the earliest and/or most

well-known CBFL measures. *Dstar*[2] and *Dstar*[3] were shown to outperform 38 existing metrics [31]. The last 5 metrics were theoretically proven to be *maximal* (i.e., most effective) [30].

The following entities need to be first computed in order to compute the suspiciousness score of a profile element:

- a_{ep}: the number of passing test cases that execute the profile element.
- a_{ef}: the number of failing test cases that execute the profile element.
- a_{np}: the number of passing test cases that do no execute the profile element.
- a_{nf}: the number of failing test cases that do not execute the profile element.
- P: the total number of passing test cases.
- F: the total number of failing test cases.

Table 1 shows the suspiciousness metrics we used in our experiments. Note how some metrics use all four of the above entities and others use part of them. For example, *Tarantula* use all four, whereas *Wong1* and *Binary* only use a_{ef}.

C. Ranking approach

After computing the suspiciousness metrics for all the collected *def-use* pairs, we assign to each Java line of code a "collective" suspiciousness score which is equal to the maximum score assigned to the *def-use* pairs traversing it. The lines of code are then ranked in decreasing order of collective suspiciousness values. We quantify the fault localization effectiveness using the EXAM score [31] which measures the percentage of lines that are ranked higher than the faulty one. As such, lower EXAM scores indicate better performance. In case the faulty line is tied with other lines in terms of collective suspiciousness score, we rank it in the middle of them. Note that even though our underlying analysis is conducted at the Java bytecode level, the ranking is conducted at the Java line level; the reason stems from the fact that it is easier to define a bug location in terms of line number as opposed to byte code offset.

III. Experimental Evaluation

A. Subject Programs

Our experiments involved 20 defective versions from the *Siemens* benchmark (*sir.unl.edu*) that we previously converted to Java [1]; specifically, two versions from *print_tokens*, and three from each of the remaining six versions. Defects involving missing code were omitted since our ranking approach would not be applicable in such cases.

For a given test suite, the test cases were originally classified as failing or passing. As part of previous work [19], the passing tests were further categorized as *true passing*, *weak CC*, or *strong CC*. This was done by manually inserting two code checkers near the fault: 1) a checker that detects weak CC tests by monitoring whether the fault was reached; and 2) a checker that detects strong CC tests by monitoring whether the fault was reached and the program has transitioned into an infectious state (the condition that captures whether an infection occurred, is mostly inferred by comparing the faulty code and its corresponding fix).

Suspiciousness Metrics	**Expression**
Tarantula [12][13]	$\frac{a_{ef}}{a_{ef}+a_{nf}} / (\frac{a_{ef}}{a_{ef}+a_{nf}} + \frac{a_{ep}}{a_{ep}+a_{np}})$
Ample [7]	$\frac{a_{ef}}{a_{ef}+a_{nf}} - \frac{a_{ep}}{a_{ep}+a_{np}}$
Ochiai [2]	$\frac{a_{ef}}{\sqrt{(a_{ef}+a_{nf})(a_{ef}+a_{ep})}}$
$Dstar^2$ [29]	$\frac{(a_{ef})^2}{a_{nf}+a_{ep}}$
$Dstar^3$ [29]	$\frac{(a_{ef})^3}{a_{nf}+a_{ep}}$
Naish1 [29]	$\begin{cases} -1 & \text{if } a_{ef} < F \\ P - a_{ep} & \text{if } a_{ef} = F \end{cases}$
Naish2 [29]	$a_{ef} - \frac{a_{ep}}{a_{ep}+a_{np}+1}$
Wong1 [29]	a_{ef}
Russel [29]	$\frac{a_{ef}}{a_{ef}+a_{nf}+a_{ep}+a_{np}}$
Binary [29]	$\begin{cases} 0 & \text{if } a_{ef} < F \\ 1 & \text{if } a_{ef} = F \end{cases}$

Table 1 – Evaluated suspiciousness metrics

B. Suspiciousness Scores Results

Table 2 and Table 3 present our results related to the suspiciousness scores. For each of the four categories of test suites, it shows the suspiciousness scores assigned to the faulty lines of code. The results are summarized by showing: 1) the average score across the 20 defects (**Avg**); 2) the average increase for $\boldsymbol{T_{CC\emptyset}}$, $\boldsymbol{T_{CCw}}$, and $\boldsymbol{T_{CCs}}$ compared to $\boldsymbol{T_{CCw+s}}$ (**AvgIncrease**); 3) the number of defects for which the score increased compared to $\boldsymbol{T_{CCw+s}}$ (**#Improved**); and 4) the number of defects for which the score was unchanged (**#Unchanged**).

For the case of *Tarantula*, *Ample*, *Ochiai*, *Dstar*[2], *Dstar*[3], *Naish2*, and *Russel*, the suspiciousness scores increased on average when the CC tests are discarded (see **Avg** and **AvgIncrease**). The average increase was largest when both weak and strong CCs were discarded ($\boldsymbol{T_{CC\emptyset}}$), less when weak CCs were discarded ($\boldsymbol{T_{CCs}}$), and least when strong CCs were discarded ($\boldsymbol{T_{CCw}}$).

For the case of *Tarantula*, *Ample*, *Ochiai*, *Dstar*[2], *Dstar*[3], and *Russel*, the suspiciousness scores of all defects increased or were unchanged (see **#Improved** and **#Unchanged**).

Naish1 is the only case where discarding the CC tests resulted in many defects to have their suspiciousness scores decrease.

Discarding CC tests had no effect on *Wong1* and *Binary*, which is expected since neither use a_{ep} nor a_{np}. Therefore, *Wong1* and *Binary* are immune to CC (a positive characteristic), however, the EXAM scores they exhibited were lower than those for the other metrics (see Tables 4 and 5).

In summary, discarding the CC tests increased the suspiciousness scores of the defects for seven metrics, decreased them for one metric, and had no effect for two metrics (that do not use a_{ep} nor a_{np}).

C. Ranking Scores Results

Since a lower EXAM score indicates better performance, Tables 4 and 5 show the average decrease for $\boldsymbol{T_{CCø}}$, $\boldsymbol{T_{CCw}}$, and $\boldsymbol{T_{CCs}}$ compared to $\boldsymbol{T_{CCw+s}}$ (**AvgDecrease**) as opposed to the average increase (**AvgIncrease**).

Discarding the CC tests had no effect on the EXAM scores of *Russel*, *Wong1*, and *Binary*; which is expected for the latter two.

For the remaining seven metrics, most of the defects were associated with a decrease in the EXAM score. The **AvgDecrease** values were negative in some cases, however, the overall **#Improved** values for these seven metrics were considerable.

In summary, discarding the CC tests had: 1) no effect on the EXAM scores of the defects in case of *Russel*, *Wong1*, and *Binary;* 2) a strong positive effect in case of *Tarantula, Ample, Ochiai, Dstar*^{2}, and *Dstar*^{3}; and 3) a mild positive effect in case of *Naish1* and *Naish2*.

D. Threats to Validity

In addition to the fact that our study is very small, there are two internal threats to the validity of our experiments:

1) The EXAM score is computed at the Java line number level as opposed to the bytecode level. This, coupled with the fact that the Siemens programs are small, might lessen the confidence of our findings in regard to the effect of CC on the EXAM scores (i.e., Tables 4 and 5). However, this concern does not apply to our findings in regard to the suspiciousness scores, since for that, the analysis is conducted at the bytecode level.
2) The EXAM score ranking is just one approach of many presented in the litterature. Other approaches should have also been used to evaluate the effect of CC on CBFL metrics.

IV. Related Work

Closely related work conducted by the authors include: 1) showing that both strong and weak CC is prevalent; 2) demonstrating that weak CC is a safety reducing factor for CBFL ; 3) and cleansing test suites from CC [17][18][19][20].

Also, in [21][22] the authors conducted an empirical study in which it was found that most dynamic dependences do not convey any measurable information, which means that it is likely that many infectious states might not propagate to the output, thus, leading to coincidental correctness.

Wang *et al* presented a coverage refinement approach [28] to reduce the influence of coincidental correctness on fault localization. The work introduces a concept called *context-pattern*, which is unique for each fault type and describes the program behavior before and after the faulty code. Coverage results for all statements are refined with the context-pattern following a context-pattern matching.

Bandyopadhyay and Ghosh considered any passing test that is similar to failing tests as CC. They proposed an iterative approach that leverages user feedback to improve the detection of CC tests and consequently CBFL [5].

Forgacs and Bertolino [9] introduced the notion of untested statements, i.e., statements that failed to exhibit any influence on the output via dynamic dependence. They attributed this phenomenon to coincidental correctness.

Hierons [10] recognized the negative effect of CC when augmenting Partition Analysis with Boundary Value Analysis. He also showed how Boundary Value Analysis can be enhanced in order to reduce the likelihood of CC even in an environment that involves non-determinism and floating point numbers.

V. Conclusions

We presented the results of our experiments that aimed at assessing the impact of coincidental correctness on the effectiveness of ten CBFL metrics. We observed that when the CC tests are discarded, in most cases the suspiciousness score of the defective statement increased and its EXAM ranking score also improved.

The metrics that benefited most from discarding CC tests are: *Tarantula, Ample, Ochiai, Dstar*^{2}*, and Dstar*^{3}. Whereas, discarding CC tests had no effect on *Russel*, *Wong1*, and *Binary*. However, the latter three metrics were the worst performers in regard to the EXAM score.

In the future, we will conduct larger experiments involving more subject programs from different domains, more metrics, and more ranking techniques.

	Tarantula				*Ample*				*Ochiai*				*Dstar2*				*Dstar3*			
	T_{CCw+s}	$T_{CC\emptyset}$	T_{CCw}	T_{CCs}	T_{CCw+s}	$T_{CC\emptyset}$	T_{CCw}	T_{CCs}	T_{CCw+s}	$T_{CC\emptyset}$	T_{CCw}	T_{CCs}	T_{CCw+s}	$T_{CC\emptyset}$	T_{CCw}	T_{CCs}	T_{CCw+s}	$T_{CC\emptyset}$	T_{CCw}	T_{CCs}
pt_tok2_v4	0.772	1	1	0.772	0.705	1	1	0.705	0.482	1	1	0.482	100.295	Infinity	Infinity	100.295	33297.88	Infinity	Infinity	33297.88
pt_tok2_v7	0.76	1	0.769	0.97	0.685	1	0.7	0.969	0.382	1	0.394	0.842	35.325	Infinity	37.987	504.106	7312.237	Infinity	7863.247	104349.9
pt_tok2_v8	0.549	1	0.549	1	0.18	1	0.18	1	0.276	1	0.276	1	21.039	Infinity	21.039	Infinity	5385.944	Infinity	5385.944	Infinity
pt_tok_v2	0.847	1	0.943	0.912	0.768	1	0.821	0.851	0.241	1	0.302	0.37	2.956	Infinity	4.799	7.613	133.029	Infinity	215.936	342.575
pt_tok_v7	0.938	1	0.938	1	0.934	1	0.934	1	0.308	1	0.308	1	2.925	Infinity	2.925	Infinity	81.91	Infinity	81.91	Infinity
repl_v16	0.831	1	0.831	0.999	0.796	1	0.797	0.999	0.262	1	0.263	0.971	6.041	Infinity	6.069	1344.8	495.389	Infinity	497.625	110273.6
repl_v7	0.786	1	0.786	0.998	0.727	1	0.728	0.998	0.23	1	0.23	0.955	4.623	Infinity	4.648	861.125	383.75	Infinity	385.821	71473.38
repl_v8	0.816	1	0.82	0.99	0.774	1	0.78	0.99	0.514	1	0.521	0.956	149.573	Infinity	154.791	4437.333	62222.38	Infinity	64392.93	1845931
sched2_v5	0.54	1	0.541	0.641	0.148	1	0.152	0.44	0.118	1	0.119	0.574	0.449	Infinity	0.462	15.754	14.359	Infinity	14.787	504.123
sched2_v6	0.519	1	1	0.519	0.073	1	1	0.073	0.053	1	1	0.053	0.02	Infinity	Infinity	0.02	0.137	Infinity	Infinity	0.137
sched2_v7	0.566	1	0.567	0.688	0.234	1	0.236	0.547	0.122	1	0.123	0.665	0.468	Infinity	0.477	24.641	14.518	Infinity	14.799	763.872
sched_v2	0.638	1	1	0.638	0.434	1	1	0.434	0.363	1	1	0.363	31.91	Infinity	Infinity	31.91	6701.158	Infinity	Infinity	6701.158
sched_v3	0.675	1	0.69	0.908	0.519	1	0.551	0.898	0.342	1	0.362	0.722	21.085	Infinity	24.009	173.158	3352.526	Infinity	3817.359	27532.05
sched_v4	0.614	1	0.668	0.708	0.371	1	0.503	0.587	0.407	1	0.504	0.568	58.363	Infinity	99.926	140.318	17158.8	Infinity	29378.25	41253.55
tcas_v1	0.887	1	0.887	1	0.873	1	0.873	1	0.643	1	0.643	1	92.263	Infinity	92.263	Infinity	12086.51	Infinity	12086.51	Infinity
tcas_v2	0.767	1	1	0.719	0.696	1	1	0.608	0.355	1	1	0.355	9.654	Infinity	Infinity	9.654	646.802	Infinity	Infinity	646.802
tcas_v5	0.573	1	0.577	0.576	0.254	1	0.268	0.263	0.092	1	0.095	0.326	0.084	Infinity	0.091	1.19	0.845	Infinity	0.909	11.905
tot_inf_v4	0.616	1	0.652	0.747	0.377	1	0.467	0.661	0.222	1	0.265	0.379	1.715	Infinity	2.486	5.528	56.594	Infinity	82.048	182.421
tot_inf_v5	0.677	1	0.635	0.996	0.523	1	0.426	0.996	0.237	1	0.237	0.967	1.723	Infinity	1.73	420.5	49.977	Infinity	50.183	12194.5
tot_inf_v8	0.967	1	0.979	0.987	0.966	1	0.979	0.987	0.934	1	0.958	0.973	1365.552	Infinity	2200.056	3600.091	271744.8	Infinity	437811.1	716418.1
Avg	**0.717**	**1**	**0.792**	**0.838**	**0.552**	**1**	**0.67**	**0.75**	**0.329**	**1**	**0.48**	**0.676**	**95.303**	**Infinity**	**165.86**	**686.943**	**21056.98**	**Infinity**	**35129.96**	**174816.3**
AvgIncrease	**-**	**0.283**	**0.095**	**0.145**	**-**	**0.448**	**0.176**	**0.264**	**-**	**0.671**	**0.314**	**0.513**	**-**	**Infinity**	**0.425**	**0.861**	**-**	**Infinity**	**0.401**	**0.88**
#Improved	**-**	**20**	**16**	**17**	**-**	**20**	**16**	**17**	**-**	**20**	**16**	**17**	**-**	**20**	**16**	**17**	**-**	**20**	**16**	**17**
#Unchanged	**-**	**0**	**4**	**3**	**-**	**0**	**4**	**3**	**-**	**0**	**4**	**3**	**-**	**0**	**4**	**3**	**-**	**0**	**4**	**3**

Table 2 – Suspiciousness scores for *Tarantula, Ample, Ochiai, Dstar[2], and Dstar[3]*

	Naish1				*Naish2*				*Wong1*				*Russel*				*Binary*			
	T_{CCw+s}	$T_{CC\emptyset}$	T_{CCw}	T_{CCs}	T_{CCw+s}	$T_{CC\emptyset}$	T_{CCw}	T_{CCs}	T_{CCw+s}	$T_{CC\emptyset}$	T_{CCw}	T_{CCs}	T_{CCw+s}	$T_{CC\emptyset}$	T_{CCw}	T_{CCs}	T_{CCw+s}	$T_{CC\emptyset}$	T_{CCw}	T_{CCs}
pt_tok2_v4	2624	2624	2624	2624	331.705	332	332	331.705	332	332	332	332	0.082	0.112	0.112	0.082	1	1	1	1
pt_tok2_v7	2635	2626	2635	2626	206.685	207	206.7	206.969	207	207	207	207	0.051	0.073	0.052	0.071	1	1	1	1
pt_tok2_v8	684	552	684	552	255.18	256	255.18	256	256	256	256	256	0.063	0.317	0.063	0.317	1	1	1	1
pt_tok_v2	2476	2476	2476	2476	47.616	48	47.69	47.851	48	48	48	48	0.012	0.019	0.013	0.016	1	1	1	1
pt_tok_v7	3774	3685	3774	3685	27.934	28	27.934	28	28	28	28	28	0.007	0.008	0.007	0.008	1	1	1	1
repl_v16	4347	3941	4347	3941	81.796	82	81.797	81.999	82	82	82	82	0.015	0.02	0.015	0.02	1	1	1	1
repl_v7	3969	3941	3969	3941	82.727	83	82.728	82.998	83	83	83	83	0.015	0.021	0.015	0.021	1	1	1	1
repl_v8	3969	3941	3969	3941	415.774	416	415.78	415.99	416	416	416	416	0.075	0.095	0.076	0.095	1	1	1	1
sched2_v5	396	50	396	51	31.148	32	31.152	31.444	32	32	32	32	0.012	0.39	0.012	0.216	1	1	1	1
sched2_v6	196	130	130	196	6.073	7	7	6.073	7	7	7	7	0.003	0.051	0.051	0.003	1	1	1	1
sched2_v7	627	43	623	47	30.234	31	30.237	30.552	31	31	31	31	0.011	0.419	0.012	0.265	1	1	1	1
sched_v2	1058	1058	1058	1058	209.434	210	210	209.434	210	210	210	210	0.079	0.166	0.166	0.079	1	1	1	1
sched_v3	1292	1292	1292	1292	158.519	159	158.551	158.899	159	159	159	159	0.06	0.11	0.063	0.1	1	1	1	1
sched_v4	875	875	875	875	293.372	294	293.503	293.587	294	294	294	294	0.111	0.251	0.145	0.165	1	1	1	1
tcas_v1	1280	1121	1280	1121	130.873	131	130.873	131	131	131	131	131	0.082	0.105	0.082	0.105	1	1	1	1
tcas_v2	1065	722	1065	722	66.696	67	67	66.609	67	67	67	67	0.042	0.085	0.059	0.053	1	1	1	1
tcas_v5	403	30	403	30	9.254	10	9.269	9.27	10	10	10	10	0.006	0.25	0.007	0.081	1	1	1	1
tot_inf_v4	384	384	384	384	32.377	33	32.468	32.662	33	33	33	33	0.031	0.079	0.039	0.054	1	1	1	1
tot_inf_v5	535	284	361	458	28.523	29	28.427	28.996	29	29	29	29	0.028	0.093	0.033	0.059	1	1	1	1
tot_inf_v8	824	824	824	824	198.966	199	198.979	198.987	199	199	199	199	0.189	0.195	0.191	0.192	1	1	1	1
Avg	**1670.65**	**1529.95**	**1658.45**	**1542.2**	**132.244**	**132.7**	**132.363**	**132.451**	**132.7**	**132.7**	**132.7**	**132.7**	**0.049**	**0.143**	**0.061**	**0.1**	**1**	**1**	**1**	**1**
AvgIncrease	**-**	**-0.092**	**-0.007**	**-0.083**	**-**	**0.003**	**0.001**	**0.002**	**-**	**0**	**0**	**0**	**-**	**0.659**	**0.197**	**0.513**	**-**	**0**	**0**	**0**
#Improved	**-**	**0**	**0**	**0**	**-**	**20**	**16**	**16**	**-**	**0**	**0**	**0**	**-**	**20**	**14**	**17**	**-**	**0**	**0**	**0**
#Unchanged	**-**	**7**	**17**	**8**	**-**	**0**	**3**	**3**	**-**	**20**	**20**	**20**	**-**	**0**	**6**	**3**	**-**	**20**	**20**	**20**

Table 3 – Suspiciousness scores for *Naish1, Naih2, Wong1, Russel, Binary.*

	Tarantula				*Ample*				*Ochiai*				*Dstar2*				*Dstar3*			
	T_{CCw+s}	$T_{CC\emptyset}$	T_{CCw}	T_{CCs}	T_{CCw+s}	$T_{CC\emptyset}$	T_{CCw}	T_{CCs}	T_{CCw+s}	$T_{CC\emptyset}$	T_{CCw}	T_{CCs}	T_{CCw+s}	$T_{CC\emptyset}$	T_{CCw}	T_{CCs}	T_{CCw+s}	$T_{CC\emptyset}$	T_{CCw}	T_{CCs}
pt_tok2_v4	0.161	0.044	0.044	0.161	0	0	0	0	0.051	0	0	0.051	0.051	0	0	0.051	0.036	0	0	0.036
pt_tok2_v7	0.088	0.022	0.088	0.066	0.022	0.015	0.022	0.015	0.022	0.015	0.022	0.015	0.022	0.015	0.022	0.015	0.022	0.015	0.022	0.015
pt_tok2_v8	0.526	0.109	0.526	0.109	0.489	0.015	0.489	0.015	0.336	0.015	0.336	0.015	0.285	0.015	0.285	0.015	0.248	0.015	0.248	0.015
pt_tok_v2	0.067	0.095	0.106	0.05	0.028	0.034	0.028	0.05	0.028	0.034	0.039	0.006	0.028	0.034	0.028	0.006	0.028	0.034	0.028	0.006
pt_tok_v7	0.039	0.073	0.039	0.073	0.006	0.073	0.006	0.073	0.006	0.073	0.006	0.073	0.006	0.073	0.006	0.073	0.006	0.073	0.006	0.073
repl_v16	0.049	0.063	0.049	0.092	0.049	0.035	0.049	0.035	0.049	0.035	0.049	0.035	0.049	0.035	0.049	0.035	0.049	0.035	0.049	0.035
repl_v7	0.049	0.063	0.049	0.021	0.049	0.021	0.049	0.021	0.049	0.021	0.049	0.021	0.049	0.021	0.049	0.021	0.049	0.021	0.049	0.021
repl_v8	0.085	0.049	0.085	0.049	0.063	0.049	0.063	0.049	0.07	0.049	0.07	0.049	0.063	0.049	0.07	0.049	0.063	0.049	0.063	0.049
sched2_v5	0.33	0.22	0.3	0.559	0.313	0.154	0.3	0.476	0.203	0.154	0.189	0.471	0.203	0.154	0.189	0.471	0.194	0.154	0.172	0.405
sched2_v6	0.447	0.381	0.381	0.447	0.416	0.142	0.142	0.416	0.376	0.142	0.142	0.376	0.376	0.142	0.142	0.376	0.31	0.142	0.142	0.31
sched2_v7	0.208	0.434	0.204	0.429	0.119	0.332	0.115	0.217	0.111	0.332	0.106	0.217	0.111	0.332	0.106	0.217	0.111	0.332	0.106	0.199
sched_v2	0.297	0.11	0.11	0.297	0.198	0.064	0.064	0.198	0.18	0.064	0.064	0.18	0.18	0.064	0.064	0.18	0.169	0.064	0.064	0.169
sched_v3	0.163	0.105	0.163	0.203	0.064	0.023	0.052	0.023	0.07	0.023	0.081	0.035	0.07	0.023	0.081	0.035	0.052	0.023	0.052	0.023
sched_v4	0.297	0.116	0.297	0.25	0.238	0.012	0.174	0.017	0.203	0.012	0.18	0.017	0.203	0.012	0.169	0.017	0.081	0.012	0.081	0.012
tcas_v1	0.059	0.074	0.059	0.074	0.059	0.074	0.059	0.074	0.059	0.074	0.059	0.074	0.059	0.074	0.059	0.074	0.059	0.074	0.059	0.074
tcas_v2	0.059	0.176	0	0.25	0	0.103	0	0.162	0	0.103	0	0.162	0	0.103	0	0.162	0	0.103	0	0.162
tcas_v5	0.426	0.338	0.426	0.559	0.25	0.191	0.25	0.5	0.221	0.191	0.221	0.397	0.221	0.191	0.221	0.382	0.162	0.191	0.162	0.353
tot_inf_v4	0.297	0.043	0.267	0.172	0.155	0.043	0.103	0.052	0.155	0.043	0.116	0.095	0.155	0.043	0.116	0.095	0.125	0.043	0.095	0.052
tot_inf_v5	0.216	0.156	0.299	0.009	0.069	0.043	0.108	0.009	0.078	0.043	0.152	0.009	0.078	0.043	0.108	0.009	0.069	0.043	0.087	0.009
tot_inf_v8	0.052	0.043	0.091	0.03	0.03	0.022	0.03	0.03	0.03	0.022	0.03	0.03	0.03	0.022	0.03	0.03	0.03	0.022	0.03	0.03
Avg	**0.196**	**0.136**	**0.179**	**0.195**	**0.131**	**0.072**	**0.105**	**0.122**	**0.115**	**0.072**	**0.096**	**0.116**	**0.112**	**0.072**	**0.09**	**0.116**	**0.093**	**0.072**	**0.076**	**0.102**
AvgDecrease	-	**0.306**	**0.085**	**0.004**	-	**0.449**	**0.196**	**0.072**	-	**0.372**	**0.168**	**-0.013**	-	**0.355**	**0.198**	**-0.033**	-	**0.225**	**0.187**	**-0.098**
#Improved	-	**13**	**7**	**9**	-	**14**	**7**	**9**	-	**15**	**7**	**10**	-	**15**	**7**	**10**	-	**14**	**6**	**10**
#Unchanged	-	**0**	**10**	**3**	-	**1**	**12**	**4**	-	**0**	**10**	**4**	-	**0**	**10**	**4**	-	**0**	**13**	**4**

Table 4 – EXAM scores for *Tarantula*, *Ample*, *Ochiai*, *Dstar*[2], *and Dstar*[3]

	Naish1				*Naish2*				*Wong1*				*Russel*				*Binary*			
	T_{CCw+s}	$T_{CC\emptyset}$	T_{CCw}	T_{CCs}	T_{CCw+s}	$T_{CC\emptyset}$	T_{CCw}	T_{CCs}	T_{CCw+s}	$T_{CC\emptyset}$	T_{CCw}	T_{CCs}	T_{CCw+s}	$T_{CC\emptyset}$	T_{CCw}	T_{CCs}	T_{CCw+s}	$T_{CC\emptyset}$	T_{CCw}	T_{CCs}
pt_tok2_v4	0	0	0	0	0	0	0	0	0.299	0.299	0.299	0.299	0.299	0.299	0.299	0.299	0.299	0.299	0.299	0.299
pt_tok2_v7	0.022	0.015	0.022	0.015	0.022	0.015	0.022	0.015	0.307	0.307	0.307	0.307	0.307	0.307	0.307	0.307	0.307	0.307	0.307	0.307
pt_tok2_v8	0.029	0.015	0.029	0.015	0.029	0.015	0.029	0.015	0.307	0.307	0.307	0.307	0.307	0.307	0.307	0.307	0.307	0.307	0.307	0.307
pt_tok_v2	0.067	0.034	0.067	0.05	0.067	0.034	0.067	0.05	0.173	0.173	0.173	0.173	0.173	0.173	0.173	0.173	0.173	0.173	0.173	0.173
pt_tok_v7	0.006	0.073	0.006	0.073	0.006	0.073	0.006	0.073	0.292	0.292	0.292	0.292	0.292	0.292	0.292	0.292	0.292	0.292	0.292	0.292
repl_v16	0.049	0.035	0.049	0.035	0.049	0.035	0.049	0.035	0.162	0.162	0.162	0.162	0.162	0.162	0.162	0.162	0.162	0.162	0.162	0.162
repl_v7	0	0.021	0	0.021	0	0.021	0	0.021	0.141	0.141	0.141	0.141	0.141	0.141	0.141	0.141	0.141	0.141	0.141	0.141
repl_v8	0.063	0.049	0.063	0.049	0.063	0.049	0.063	0.049	0.19	0.19	0.19	0.19	0.19	0.19	0.19	0.19	0.19	0.19	0.19	0.19
sched2_v5	0.159	0.154	0.145	0.273	0.159	0.154	0.145	0.273	0.313	0.313	0.313	0.313	0.313	0.313	0.313	0.313	0.313	0.313	0.313	0.313
sched2_v6	0.199	0.142	0.142	0.199	0.199	0.142	0.142	0.199	0.292	0.292	0.292	0.292	0.292	0.292	0.292	0.292	0.292	0.292	0.292	0.292
sched2_v7	0.08	0.332	0.075	0.173	0.08	0.332	0.075	0.173	0.358	0.358	0.358	0.358	0.358	0.358	0.358	0.358	0.358	0.358	0.358	0.358
sched_v2	0.128	0.064	0.064	0.128	0.128	0.064	0.064	0.128	0.372	0.372	0.372	0.372	0.372	0.372	0.372	0.372	0.372	0.372	0.372	0.372
sched_v3	0.047	0.023	0.047	0.023	0.047	0.023	0.047	0.023	0.355	0.355	0.355	0.355	0.355	0.355	0.355	0.355	0.355	0.355	0.355	0.355
sched_v4	0.012	0.012	0.012	0.012	0.012	0.012	0.012	0.012	0.355	0.355	0.355	0.355	0.355	0.355	0.355	0.355	0.355	0.355	0.355	0.355
tcas_v1	0.059	0.074	0.059	0.074	0.059	0.074	0.059	0.074	0.397	0.397	0.397	0.397	0.397	0.397	0.397	0.397	0.397	0.397	0.397	0.397
tcas_v2	0	0.103	0	0.162	0	0.103	0	0.162	0.353	0.353	0.353	0.353	0.353	0.353	0.353	0.353	0.353	0.353	0.353	0.353
tcas_v5	0.132	0.191	0.132	0.265	0.132	0.191	0.132	0.265	0.294	0.294	0.294	0.294	0.294	0.294	0.294	0.294	0.294	0.294	0.294	0.294
tot_inf_v4	0.052	0.043	0.043	0.043	0.052	0.043	0.043	0.043	0.375	0.375	0.375	0.375	0.375	0.375	0.375	0.375	0.375	0.375	0.375	0.375
tot_inf_v5	0.013	0.043	0.022	0.009	0.013	0.043	0.022	0.009	0.325	0.325	0.325	0.325	0.325	0.325	0.325	0.325	0.325	0.325	0.325	0.325
tot_inf_v8	0.03	0.022	0.03	0.03	0.03	0.022	0.03	0.03	0.345	0.345	0.345	0.345	0.345	0.345	0.345	0.345	0.345	0.345	0.345	0.345
Avg	**0.057**	**0.072**	**0.05**	**0.082**	**0.057**	**0.072**	**0.05**	**0.082**	**0.3**	**0.3**	**0.3**	**0.3**	**0.3**	**0.3**	**0.3**	**0.3**	**0.3**	**0.3**	**0.3**	**0.3**
AvgDecrease	-	**-0.26**	**0.121**	**-0.44**	-	**-0.26**	**0.121**	**-0.438**	-	**0**	**0**	**0**	-	**0**	**0**	**0**	-	**0**	**0**	**0**
#Improved	-	**11**	**5**	**8**	-	**11**	**5**	**8**	-	**0**	**0**	**0**	-	**0**	**0**	**0**	-	**0**	**0**	**0**
#Unchanged	-	**2**	**14**	**5**	-	**2**	**14**	**5**	-	**20**	**20**	**20**	-	**20**	**20**	**20**	-	**20**	**20**	**20**

Table 5 – EXAM scores for *Naish1*, *Naih2*, *Wong1*, *Russel*, *Binary*.

REFERENCES

[1] Rawad Abou Assi, Wes Masri. Identifying Failure-Correlated Dependence Chains. ICST Workshops 2011: 607-616.

[2] Abreu R., Zoeteweij P. and Van Gemund A. J. C. On the Accuracy of Spectrum-based Fault Localization. TAIC-PART, pp. 89-98, 2007.

[3] Ammann P. and Offutt J. Introduction to Software Testing. Cambridge University Press, 2008.

[4] Thomas Ball, Mayur Naik, Sriram K. Rajamani. From symptom to cause: localizing errors in counterexample traces. POPL 2003: 97-105.

[5] Aritra Bandyopadhyay, Sudipto Ghosh. Tester Feedback Driven Fault Localization. ICST 2012: 41-50.

[6] B. Baudry, F. Fleurey, and Y. Le Traon, Improving test suites for efficient fault localization, In Proc.of ICSE'06, pages 82- 91, May, 2006.

[7] Dallmeier V., Lindig C., and Zeller A. 2005.b. Lightweight defect localization for Java. In A. P. Black, editor, ECOOP 2005: 19th European Conference, Glasgow, UK, July 25–29, 2005. Proceedings, volume 3568 of LNCS, pages 528–550. Springer-Verlag.

[8] DeMillo, R. A., Lipton, R. J., and Sayward, F. G. 1978. Hints on Test Data Selection: Help for the Practicing Programmer. Computer 11, 4 (Apr. 1978), 34-41.

[9] István Forgács, Antonia Bertolino. Preventing untestedness in data-flow based testing. Softw. Test., Verif. Reliab. 12(1): 29-58 (2002).

[10] R. M. Hierons. Avoiding coincidental correctness in boundary value analysis. ACM Transactions on Software Engineering and Methodology. Volume 15, Issue 3 (July 2006). Pages: 227 - 241.

[11] Hyunsook Do, Sebastian Elbaum, and Gregg Rothermel. Supporting Controlled Experimentation with Testing Techniques: An Infrastructure and its Potential Impact. *Empirical Software Engineering: An International Journal*, Volume 10, No. 4, pages 405-435, 2005.

[12] Jones J. and Harrold M. J. "Empirical Evaluation of the Tarantula Automatic Fault-Localization Technique," Proc. 20th IEEE/ACM Int'l Conf. Automated Software Eng. (ASE '05), pp. 273-282, 2005.

[13] Jones J., Harrold M. J., and Stasko J.. Visualization of Test Information to Assist Fault Localization. In *Proceedings of the 24th International Conference on Software Engineering*, pp. 467-477, May 2001.

[14] Liblit B., Aiken A., Zheng A., and Jordan M. Bug Isolation via Remote Program Sampling. Proc. ACM SIGPLAN 2003 Int'l Conf. Programming Language Design and Implementation (PLDI '03), pp. 141-154, 2003.

[15] Liblit B., Naik M., Zheng A., Aiken A., and Jordan M. Scalable Statistical Bug Isolation. Proc. ACM SIGPLAN 2005 Int'l Conf. Programming Language Design and Implementation (PLDI '05), pp. 15-26, 2005.

[16] Wes Masri. Fault localization based on information flow coverage. Softw. Test., Verif. Reliab. 20(2): 121-147 (2010).

[17] Wes Masri, Rawad Abou Assi. Cleansing Test Suites from Coincidental Correctness to Enhance Fault-Localization. ICST 2010: 165-174.

[18] Wes Masri, Rawad Abou Assi. Prevalence of coincidental correctness and mitigation of its impact on fault localization. ACM Trans. Softw. Eng. Methodol. 23(1): 8:1-8:28 (2014).

[19] Masri W., Abou-Assi R., El-Ghali M., and Fatairi N. An Empirical Study of the Factors that Reduce the Effectiveness of Coverage-based Fault Localization. International Workshop on Defects in Large Software Systems, DEFECTS, Chicago, IL, 2009.

[20] Wes Masri, Rawad Abou Assi, Fadi A. Zaraket, Nour Fatairi. Enhancing Fault Localization via Multivariate Visualization. ICST 2012: 737-741.

[21] Masri W., Podgurski A. 2006. An Empirical Study of the Strength of Information Flows in Programs. Fourth International Workshop on Dynamic Analysis (WODA 2006), Shanghai, China, May 2006.

[22] Masri, W. and Podgurski, A. Measuring the Strength of Information Flows in Programs. ACM Transactions on Software Engineering and Methodology (TOSEM). To appear 2009.

[23] W. Masri, A. Podgurski and D. Leon. "An Empirical Study of Test Case Filtering Techniques Based On Exercising Information Flows". IEEE Transactions on Software Engineering, July, 2007, vol. 33, number 7, page 454.

[24] Masri, W. and Podgurski, A. Algorithms and Tool Support for Dynamic Information Flow Analysis. Information and Software Technology (Elsevier), Vol. 51, Fe. 2009, pp. 395-404.

[25] W. Masri, A. Podgurski and D. Leon. "An Empirical Study of Test Case Filtering Techniques Based On Exercising Information Flows". IEEE Transactions on Software Engineering, July, 2007, vol. 33, number 7, page 454.

[26] Renieris M. and Reiss S. Fault localization with nearest-neighbor queries. In Proceedings of the 18th IEEE Conference on Automated Software Engineering, pp. 30-39, 2003.

[27] Jeffrey M. Voas: PIE: A Dynamic Failure-Based Technique. IEEE Trans. Software Eng. 18(8): 717-727 (1992).

[28] Wang X., Cheung S.C., Chan W.K., Zhang Z. Taming coincidental correctness: Coverage refinement with context patterns to improve fault localization. IEEE 31st International Conference on Software Engineering, pp. 45-55, 2009

[29] Naish L., Lee H.J., and Ramamohanarao K. A model for spectra-based software diagnosis. ACM Transactions on Software Engineering and Methodology. 20, 3, Article 11 (August 2011).

[30] X. Xie, T. Y. Chen, F.-C. Kuo, and B. Xu. A theoretical analysis of the risk evaluation formulas for spectrum-based fault localization. ACM Transactions on Software Engineering and Methodology (TOSEM), 22(4):31, 2013.

[31] W. E. Wong, V. Debroy, R. Gao, and Y. Li, "The DStar Method for Effective Software Fault Localization," IEEE Transactions on Reliability, vol. 63, no. 1, pp. 290–308, 2014.