JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 38, 357-374 (2022)
DOI: 10.6688/JISE.202203_38(2).0005

Evaluating the Influence on Fault Localization Caused by Test Suite Reduction in Continuous Integration Process*

JUTARPORN INTASARA AND CHU-TI LIN+
Department of Computer Science and Information Engineering
National Chiayi University
Chiayi, 600 Taiwan
E-mail: {s1040464; chutilin+}@mail.ncyu.edu.tw

Test suite reduction (TSR) is a frequently adopted approach to improve the efficiency of regression testing while spectrum-based fault localization (SBFL) is a famous approach to shorten the tedious debugging process. Both approaches can be incorporated into the continuous integration (CI) environment. A configuration for a CI is relevant to the settings of three TSR factors (namely the coverage granularity level, the metric to evaluate test cases, and the TSR strategy) and the selection of SBFL technique. Since different TSR techniques will produce different test logs and test logs are the inputs to SBFL techniques, the selections of TSR and SBFL techniques both impact the effectiveness of fault localization (FL). Thus, it is important for software developers to choose a suitable TSR technique and determine which SBFL technique will assort well with it. This paper also aims to investigate how each of the aforementioned parameters in a CI configuration affects FL effectiveness. Our experiment results indicate that applying TSR may be harmful to FL effectiveness and the Jaccard SBFL technique is the most effective to locate faults no matter which of the TSR techniques (including using the original test suite directly) is adopted to run regression testing. Additionally, the experimental results also indicate that it is better to perform TSR based on branch coverage information if TSR must be included in the CI. In comparison with coverage granularity level, the other two TSR factors do not cause statistically significant impact on FL effectiveness. Our findings should be useful for software developers to configure their CI.

Keywords: software testing, debugging, continuous integration, test suite reduction, fault localization

1. INTRODUCTION

In software development process, developers may frequently modify the source code in order to upgrade the system or fix bugs. The modified software needs to be validated before being released to ensure that it still works properly [1, 2], which is called regression testing. Regression testing may execute all test cases that have been run for the previous version together with new test cases that are designed for validating the new/modified functionalities in the current version [3], *i.e.*, retest-all. This means that the size of test suite may increase as software under test (SUT) evolves. Yet, it is sometimes unsuitable to run all test cases because time and cost are generally limited in practice.

In the literature, test suite reduction (TSR) (*e.g.*, [4-7]) is a frequently adopted approach to decrease the cost of regression testing. More specifically, TSR techniques aim

Received November 7, 2020; revised December 9, 2020; accepted December 30, 2020.
Communicated by Nai-Wei Lin.
+ Corresponding author.
* This work was supported by Ministry of Science and Technology Taiwan, under Grants MOST 108-2628-E-415-001-MY2 and the preliminary version of this work has been presented in the 11th International Conference on Advances in Databases, Knowledge, and Data Applications, Athens, Greece, May 2019.

to reduce the size of test suite by removing the redundant test cases; on the other hand, software debugging [8, 9] is an important activity in software development and maintenance. This activity attempts to pinpoint the locations of faults, namely fault localization (FL), and then design and implement the fixes. Software developers usually remove faults manually [10], thus resulting in a time-consuming, tedious, and expensive debugging process. An automated debugging approach can address this problem and make software more reliable and maintainable [11]. According to the classification described in [12], the existing FL techniques can be classified into eight categories. Spectrum-based fault localization (SBFL) achieves significant result with low overheads in comparison to the other types of approaches [13]. Thus, we focus our study on SBFL in this paper.

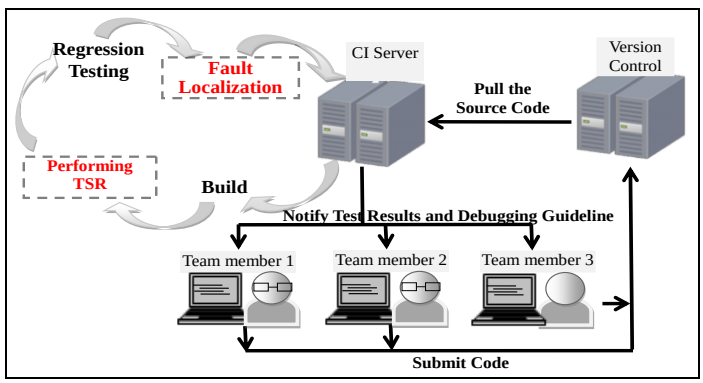

Fig. 1. The CI process enhanced by including TSR and FL techniques.

Continuous integration (CI) [14, 15] is a software development strategy in which software developers frequently and regularly integrate the new or modified code to the repository of source code. When a part of the program is changed/added and submitted to the repository, the modifications should be tested in order to ensure that they will not result in unexpected side effects. CI may become more desirable if TSR techniques and FL techniques are included. For example, before regression testing, software developers can use a TSR technique to choose test cases so as to render the regression testing more cost-effective. After testing, software developers can apply a SBFL technique to accelerate the fault removal. That is, after developers submit the modified modules to codebase, TSR techniques can be used together with SBFL techniques to decrease the time taken for testing and debugging. Fig. 1 shows an example of the enhanced CI process. Yet, different TSR techniques produce different test logs (*e.g.*, revealed faults and code coverage). Furthermore, since the test logs are the inputs to SBFL techniques, the locations of faults that are identified by SBFL techniques based on different test logs will be significantly diverse. Thus, the selection of TSR technique has observable consequences on the effectiveness of regression testing, and further has observable consequences on the FL effectiveness. In the literature, some studies (*e.g.*, [8, 9, 16]) aim to propose TSR techniques so as to produce a subset of test cases that can achieve better FL effectiveness than the original test suite (OTS).

Actually, the FL effectiveness is relevant to the adopted test suite and the source code of the SUT. Since there exist a lot of TSR and SBFL techniques in the literature, it is

difficult for software developers to choose a suitable TSR technique and determine which SBFL technique will assort well with it. In our preliminary study [17], we have described this tough problem and this study aims to tackle it. In Section 2, we will discuss 3 TSR factors that may affect FL effectiveness and then define the CI configuration in our work. In addition to the created test suites, this study will cover OTS, 36 TSR techniques, and 4 SBFL techniques, resulting in (1+36)×4=148 CI configurations. We will empirically evaluate the FL effectiveness achieved by them.

The remainder of this paper is organized as follows. Section 2 will give the necessities for developing our empirical study (*i.e.*, the fundamentals of TSR and SBFL). Section 3 will outline our research questions, describe the criterion used to evaluate FL effectiveness in our study, and the experimental setup. Section 4 will answer the research questions by reporting and discussing the experimental results and highlight our findings. Section 5 will discuss the related work. Finally, Section 6 will show the concluding remarks.

2. BACKGROUND

2.1 TSR Techniques

TSR [18, 19] aims to produce a subset of the OTS, called the representative set (RS), by repeatedly removing the redundant test cases according to specific rules until all test requirements are satisfied. Thus, the TSR problem can be defined as follows:

Given: • An original test suite $OTS = \{t_1, t_2, t_3, ..., t_m\}$, where t_i (i = 1 to m) represents the ith test case in the test suite.
• A set of test requirements $REQ = \{r_1, r_2, r_3, ..., r_n\}$, where r_j (j = 1 to n) represents the jth test requirement.
• A binary relation $COV = \{(t, r) \mid t \in OTS, r \in REQ$, and the test case t can satisfy the test requirement $r\}$.

Objective: Find a representative set *RS*, which is a subset of *OTS*, in which the test cases can satisfy all test requirements satisfied by *OTS*.

The coverage granularity level, the metric to evaluate test cases (hereafter called reduction metric for simplicity) and the TSR strategy are the important factors to design a TSR technique. That is, a TSR technique is defined as a combination of a TSR strategy, a reduction metric, and a coverage granularity level. Section 2.1.1 will review the coverage granularity, while Sections 2.1.2 and 2.1.3 will review the reduction metrics and several well-known TSR strategies, respectively.

2.1.1 Review of code coverage granularity

In the literature, many studies (*e.g.*, [20-23]) regard the test requirement as a code entity (*e.g.*, a code statement, a branch, or a function). "Satisfying test requirements" indicates the program path covers a set of code entities when running test cases. For example, if the coverage granularity level adopted by software developers to perform TSR is statement, the test cases in the RS should cover all code statements that are covered by those in

the OTS. Because performing TSR at different coverage granularities will produce different RS and then impact the consequent SBFL, we also consider coverage granularity as an important factor for TSR.

2.1.2 Review of reduction metrics

In [20], Lin *et al.* adopted 4 reduction metrics, namely Coverage, Ratio, Irreplaceability, and EIrreplaceability, to evaluate the importance of test cases during TSR from different standpoints. The descriptions of these 4 metrics are given as follows.

Coverage: This metric indicates the number of test requirements (*e.g.*, code statements, branches, or functions in the SUT) that are covered by test cases and it can be considered a criterion to assess the completion of a test [24]. Many code-based TSR techniques (*e.g.*, [21, 23, 25]) adopt this metric (hereafter called Cov) to evaluate test cases.

Ratio: Lin *et al.* [20] indicated that there may be significant difference in execution time among test cases. Thus, minimizing the size of test suite may not minimize the time taken to run test cases. Ma *et al.* [26] and Smith and Kapfhammer [27] also suggested that, in addition to the code coverage achieved by test cases, the cost of running test cases should be considered for TSR. Thus, the studies in [26, 27] evaluate test cases according to a cost-aware metric, called Ratio hereafter in this paper.

Irreplaceability: If a test requirement that is satisfied by the test case *t* can also be satisfied by many of the other test cases, the probability of that test requirement being still covered by the other test cases should be high even though *t* is not included in the RS. In other words, the test cases that satisfy such test requirements can be regarded as replaceable. In contrast, the irreplaceability of a test case is high if it is not easy to find the other test cases to replace it in order to satisfy a specific set of test requirements. Based on the concept of irreplaceability and the cost of running each individual test case, Lin *et al.* [20] proposed a cost-aware metric called Irreplaceability (hereafter called Irre for simplicity) to evaluate test cases' contribution on increasing code coverage.

EIrreplaceability: Since the essential test case should be included in the RS as early as possible to avoid choosing redundant test cases, Lin *et al.* [20] also considered the essentials strategy and posited that the irreplaceability of essential test cases should be given the highest priority. They further proposed an enhancement of Irre, called EIrreplaceability (hereafter called EIrre for simplicity).

2.1.3 Review of TSR strategies

In the literature, a lot of TSR strategies have been proposed. In order to avoid blurring the scope of this work, this review is focused on the 3 well-known TSR strategies that will be empirically compared in our study.

Additional Greedy: The Additional Greedy algorithm [28], hereafter called AG, takes into account the number of test requirements that are satisfied by each test case during the

reduction process. Yet, AG only considers the test requirements that are not yet satisfied by the test cases in the RS when computing the coverage achieved by test cases. In other words, AG will repeatedly choose the test case that can satisfy the most unsatisfied test requirements until all test requirements are satisfied.

GRE: Chen and Lau [29] proposed a heuristic, called GRE, based on 3 strategies including: (1) the essentials strategy: first select the test cases that satisfy test requirements that cannot be satisfied by the others; (2) the 1-to-1 redundancy strategy: remove the test case if all requirements satisfied by it can also be satisfied by another in the RS; and (3) the Greedy strategy: perform the AG algorithm. Some essential test cases may appear after applying the 1-to-1 strategy, and vice versa. Thus, the essentials and the 1-to-1 redundancy strategies are complementary to each other and should be alternately applied. The AG algorithm will be applied if both of the other two strategies cannot be applied.

HGS: This strategy is called HGS [30] due to the abbreviations of the authors' last names, Harrold, Gupta, and Soffa. Let S_j (for $j = 1, 2, 3, \ldots, n$) represent the subsets of the OTS, with each subset S_j including all unselected test cases in the OTS that can satisfy the jth test requirement. This strategy will start from the subset S_j with the smallest cardinality, select the test cases in S_j that belongs to the most subsets of the same cardinality, mark all test requirements that are satisfied by the selected test case, and remove that test case from all subsets. If a tie occurs during selecting test cases in S_j, it will recursively consider the number of unsatisfied test requirements that are satisfied by the candidates and choose the test case that can satisfy the most unsatisfied test requirements. The HGS will focus on the subset S_j with the smallest cardinality and iteratively repeat the aforementioned steps until all test requirements have been satisfied.

2.2 SBFL Techniques

Performing SBFL [9, 12] techniques is one of frequently used approaches to identify the locations of the bugs. These techniques use the test logs (including the program execution/code coverage information and the result of test cases) to calculate the probability of each code statement being faulty, called the suspiciousness, and then produce a ranking list of all code statements according to their suspiciousness in descending order. Many SBFL techniques (*e.g.*, Tarantula [31], Ochiai [32], Jaccard [32], and SBI [33]) have received considerable attention and were frequently used to be the baseline techniques for comparing with the new ones [34, 35]. The remaining of this section is dedicated to the review of these 4 SBFL techniques which are adopted in our empirical study.

Tarantula: Jones *et al.* [31] proposed the technique, called Tarantula, to pinpoint the faulty statements. The main idea underlying this technique is that the statement has a high probability to be faulty if it is executed by most of the failed test cases in the test suite. Additionally, the statements that are covered by more test cases will be assigned higher confidence values. If a tie occurs when ranking the statements based on their suspiciousness, their confidence values will be further evaluated. The statement in the tie will be examined first if it has the highest confidence value. Moreover, if the statements in the tie also have the same confidence value, these statements will then be given the same ranking according to Renieris and Reiss's approach [36].

Ochiai: Ochiai [37] proposed a metric to computing genetic similarity in molecular biology (*e.g.*, [38, 39]). Abreu *et al.* [32] and Naish *et al.* [40] adopted a variant of Ochiai's metric to calculate the suspiciousness of code statements and then sorted the statements by the similar manner that is described for Tarantula.

Jaccard: Abreu *et al.* [32] also used the Jaccard similarity coefficient which was proposed by Baudrey *et al.* [41] to compute the statements' suspiciousness when performing SBFL. Similar to Tarantula, this technique also considers confidence values if ties occur.

SBI: The Statistical Bug Isolation (SBI) was originally proposed by Liblit *et al.* [33] for computing the predicates' suspiciousness. Yu *et al.* [10] adopted it to compute the statements' suspiciousness in order to facilitate the comparison with the other SBFL techniques.

3. EXPERIMENTAL SETUP

This section will show and explain our research questions, the metric for evaluating FL effectiveness, and the statistical tests, respectively. Additionally, this section will also design the experiment in order to answer these research questions.

3.1 Research Questions

We will carry out the experiment to answer the following research questions:

RQ1: If adopting the reduced test suite instead of the original test suite, is the FL effectiveness still preserved?
RQ2: Does each factor of the CI configurations affect FL effectiveness?

In order to answer this research question, we will analyze the influence on FL effectiveness caused by adjusting each of the 4 factors in the CI configuration. Thus, RQ2 will be further divided into RQs2.1 through 2.4.

RQ2.1: Does the selection of coverage granularity level affect FL effectiveness?
RQ2.2: Does the selection of the reduction metric affect FL effectiveness?
RQ2.3: Does the selection of TSR strategy affect FL effectiveness?
RQ2.4: Does the selection of SBFL technique affect FL effectiveness?

RQ3: What is the most recommended CI configuration for optimizing FL effectiveness?

3.2 Evaluation Criterion and Statistical Tests

In our experiment, we performed TSR and SBFL based on a lot of different configurations. We adopted the criterion, *EXAM* score [11, 34, 35], to evaluate the FL effectiveness achieved by each of these configurations. The *EXAM* score represents the percentage of statements in a program that have to be examined until the first faulty statement is located. In the literature, it has been commonly used to evaluate FL effectiveness. The lower *EXAM* score is, the more effective FL is.

To confirm the comparisons on the FL effectiveness, we adopted the Kruskal-Wallis test, the Mann-Whitney *U* test, and the Vargha and Delaney test for reporting the statistical tests. We used these non-parametric tests because the assumption of one-way ANOVA is

not met in our study, *i.e.*, the *EXAM* scores are not normally distributed [42] and the null hypothesis is that the *EXAM* scores from two groups share the same distribution. First, the Kruskal-Wallis test was performed to assess whether there exists significant difference among all groups. If the significant difference is observed (*i.e.*, the *p*-value < 0.05), we further performed the two statistical tests (*i.e.*, the Mann-Whitney *U* test and the Vargha and Delaney test) for pairwise comparisons. If the *p*-value of the Mann-Whitney *U* test is less than 0.05, it indicates that the values in two groups are significantly different. The Vargha and Delaney test is used to calculate the effect size A_{12} between the values of two groups *X* and *Y*. The value of A_{12} ranges from 0 to 1. $A_{12} > 0.50$ means that *X* has higher chances to obtain a higher *EXAM* score than *Y* and vice versa. $A_{12} = 0.50$ means that two groups are equivalent in terms of *EXAM* score. Additionally, if the value of A_{12} is close to 0.50, it indicates that the difference in *EXAM* score between the two groups is small. Vargha and Delaney [43] suggested that $A_{12} > 0.64$ (or < 0.36) is indicative of "medium" effect size while $A_{12} > 0.71$ (or < 0.29) can be indicative of a promising "large" effect size.

3.3 Descriptions of Subject Programs

Our experiment uses 9 subject programs which were written in C language, namely the 7 programs in the Siemens suite (including print_tokens, print_tokens2, replace, schedule, schedule2, tcas, and tot_info), flex, and gzip. These subject programs together with the test pools are available at the Software-artifact Infrastructure Repository (SIR) [44] and are the benchmarks frequently used in prior studies (*e.g.*, [11, 16, 31, 34]). Some faulty versions were excluded in order to facilitate our experiment. For example, for some versions, no faults are revealed by all test cases, or the faults are located in the header files. We used this criterion to exclude those faulty versions in order to make our experimental results comparable to prior studies. In addition, we conducted the experiment based on flex and gzip's single-fault versions (*i.e.*, separately seeding each of the faults in each version). We totally adopted 226 faulty versions attached to these subjects and the details are described in Table 1.

Table 1. Description of the subject programs.

Subject Program	#Faulty Versions	#Used Faulty Versions	LOC[1]	Size of Test Pool[2]
print_tokens	7	3	564	4130
print_tokens2	10	8	512	4115
replace	32	24	563	5542
schedule	9	4	412	2650
schedule2	10	4	307	2710
tcas	41	37	173	1608
tot_info	23	18	406	1052
flex (v1-v5)	82	48	12421-14244	525
gzip (v1,v2,v4,&v5)	49	16	6576-7996	214

[1] It indicates the number of code statements in the subject program.
[2] It indicates the number of test cases in the test pool.

3.4 Experiment Steps

In our experiment, we created 1,000 different OTS for each faulty version, obtained an RS by performing TSR on each OTS, used both of the OTS and the RS to run regression

testing, and then performed SBFL based on the test results. The framework of our experiment is depicted in Fig. 2. In order to avoid too lengthy descriptions, the 4 reduction metrics (namely Cov, Ratio, Irre, and EIrre) will be denoted by *C*, *R*, *I*, and *E* while the 3 coverage granularity levels (*i.e.*, branch, function, and statement) will be denoted by *B*, *F*, and *S*, respectively. Thus the CI configurations that are compared in our study and the way to produce them are shown in Fig. 2. For example, *HGS-I-F* represents the TSR technique that adopts the HGS algorithm to reduce test suites according to the metric Irre to evaluate test cases' importance based on function coverage. It is noted that OTS will also be regarded as a TSR technique that removes no test cases from the test suites. Similarly, the SBFL techniques (*i.e.*, Jaccard, Ochiai, and Tarantula) will also be denoted by *J*, *O*, and *T*, respectively, and keep the abbreviation *SBI* in this figure. Additionally, *HGS-I-F-J* represents the CI configuration that adopts the TSR technique *HGS-I-F* and then locates faults using Jaccard while *OTS-T* represents the one that excludes TSR from the CI and then locates faults using Tarantula. The detailed steps of our experiment are given below.

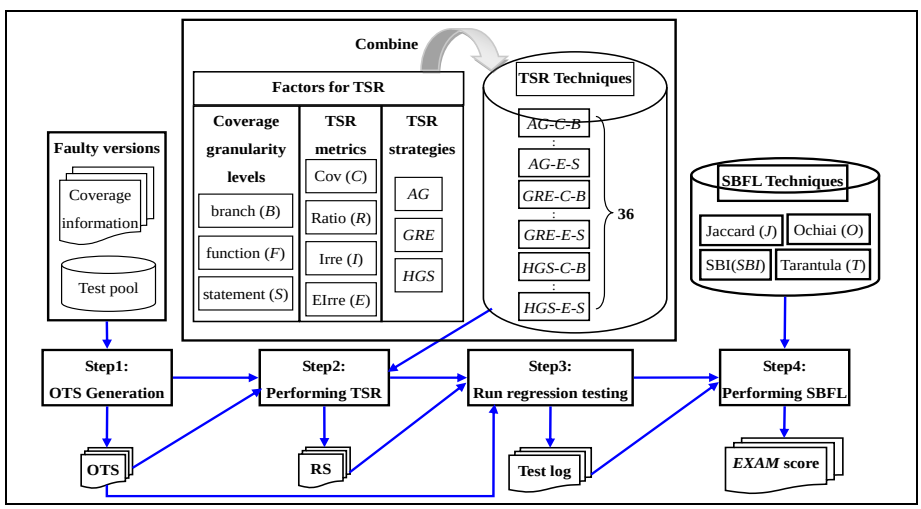

Fig. 2. The framework of our experiment.

Step 1: Create an OTS for each faulty version by randomly picking *v* test cases from the test pool, where $1 \le v \le LOC \times 0.5$ for the Siemens suite and $1 \le v \le LOC \times 0.01$ for flex and gzip. We adopted different ways to generate this random integer due to the variety of the subjects' scales. If the created OTS does not satisfy all test requirements, a new test case is randomly chosen to determine if it can cover at least one of the unsatisfied test requirements. If yes, this test case will be included in the OTS; otherwise, it will not be included in the OTS. This operation will be repeated until all test requirements are satisfied. Additionally, if the created OTS does not contain any failed test cases, we will generate a random integer v_f, where $1 \le v_f \le N_f$ and N_f represents the total number of failed test cases in the test pool, further randomly choose v_f failed test cases from the test pool, and include them in the OTS.

Step 2: Perform each of the 36 TSR techniques (*i.e.*, the combinations of 3 coverage granularities, 4 reduction metrics, and 3 TSR strategies) on OTS and obtain the RS.
Step 3: Run test cases in each OTS/RS and obtain the test logs.
Step 4: Perform each of the 4 SBFL techniques based on test logs, compute the suspiciousness, suggest the ranking list of statements, and compute the *EXAM* score.
Step 5: Repeat Steps 1-4 1,000 times to ensure the experiment's diversity and generality.

Overall, for each faulty version in our experiment, we created 1,000 OTS, 1,000×3×4×3=36,000 RS, and 1,000×(3×4×3+1)×4=148,000 *EXAM* scores (including the SBFL result based on OTS). Because 226 faulty versions are attached to the 9 subject programs, we totally created 226×1,000=226,000 OTS, 226×1,000×3×4×3=8,136,000 RS, and 226×1,000×(3×4×3+1)×4=33,448,000 *EXAM* scores. Thus, the experiment results shown in Section 4 are reported based on these 33,448,000 *EXAM* scores.

4. EXPERIMENTAL RESULTS

Sections 4.1 through 4.3 will reply to RQs 1 through 3, respectively, while Section 4.4 will summarize our findings from the experiment. Finally, Section 4.5 will discuss the threats to internal and external validity for our experiment.

4.1 Answer to RQ1: If adopting the reduced test suite instead of the original test suite, is the FL effectiveness still preserved?

Table 2 summarizes the statistics for each TSR technique in terms of *EXAM* score. Please notice that the mean and median values were computed across the *EXAM* scores produced by 4 SBFL techniques. According to this table, the *p*-value of the Kruskall-Wallis test shown in the rightmost column indicates that the null hypothesis is rejected (*i.e.*, *p*-value < 0.05). Thus, we further performed the statistical tests for pairwise comparison and showed the results in Table 3. Because there are 37×(37−1)/2=666 pairs for comparison in our study and showing all of them will take a lot of space, this table only includes *OTS* that achieves the best FL effectiveness, the two TSR techniques that achieve the second and third best (*i.e.*, *AG-I-B* and *AG-I-S*), and the two TSR techniques that achieve the worst result (*i.e.*, *GRE-E-F* and *GRE-C-F*). As seen from this table, *OTS* significantly outperforms the others since the effect size A_{12} and the *p*-value in each cell of the 2nd row are less than 0.50 and 0.05, respectively, thus indicating that locating faults based on RS will lead to significantly worse effectiveness in comparison with that based on OTS. Even though there is a significant difference in *EXAM* score between *OTS* and *AG-I-B*, A_{12} for this pair is close to 0.50, which is indicative of "small" effect size. Yet, the values of A_{12} for the pair of *OTS* and *GRE-E-F* and the pair of *OTS* and *GRE-C-F* are 0.10, thus indicating a promising "large" effect size. This phenomenon is consistent with our finding from Table 2 (*i.e.*, the *EXAM* score of *AG-I-B* is close to that of *OTS* and is clearly better than those of *GRE-E-F* and *GRE-C-F*). As such, reducing test suites based on a suitable TSR technique is important for efficient debugging.

In general, TSR is not recommended in order to maximize FL effectiveness unless software developers carefully analyze the influence caused by different coverage granu-

larity levels, reduction metrics, and TSR strategies and then determine the most suitable technique. This finding motives us to further investigate the next research question.

4.2 Answer to RQ2: Does each factor of the CI configurations affect FL effectiveness?

Since RQ2 aims to study the influence on FL effectiveness caused by each of the 4 factors in a CI configuration (*i.e.*, the 3 TSR factors and the selection of SBFL technique), we will reply the result for each of factors in Sections 4.2.1 through 4.2.4, respectively.

Table 2. The statistics in terms of *EXAM* score for *OTS* and 36 TSR techniques.

Statistics	*EXAM* score (%)													*p*-value[3]
	TSR Techniques													
	OTS	Algo[1] & Gra[2] Metric	*AG*			Algo & Gra Metric	*GRE*			Algo & Gra Metric	*HGS*			
			B	*F*	*S*		*B*	*F*	*S*		*B*	*F*	*S*	
Mean	13.16	*C*	19.01	38.47	23.44	*C*	18.95	40.65	23.52	*C*	18.59	39.09	23.15	0.00
		R	18.11	39.18	22.33	*R*	18.74	40.12	23.00	*R*	18.13	38.24	22.25	
		I	13.71	35.93	16.14	*I*	18.54	39.40	22.88	*I*	18.34	39.04	22.92	
		E	18.67	39.45	22.84	*E*	18.88	40.31	23.11	*E*	18.13	38.24	22.25	
Median	6.37	*C*	15.98	39.84	21.73	*C*	15.85	40.27	22.19	*C*	15.09	40.18	21.77	
		R	15.31	38.70	20.91	*R*	15.61	40.27	21.49	*R*	14.66	39.35	20.87	
		I	8.99	34.81	12.21	*I*	15.13	40.24	21.35	*I*	14.88	40.16	21.26	
		E	15.78	39.27	21.71	*E*	15.87	40.28	21.68	*E*	14.66	39.35	20.87	

[1] It means the TSR algorithm. [2] It means the coverage granularity level. [3] It indicates the statistical result of the Kruskal-Wallis test.

Table 3. The pairwise comparisons for RQ1.

Pairwise TSR techniques	*AG-I-B*	*AG-I-S*	*GRE-E-F*	*GRE-C-F*
OTS	0.46/0.00	0.37/0.00	0.10/0.00	0.10/0.00
AG-I-B	–	0.40/0.00	0.11/0.00	0.10/0.00
AG-I-S	–	–	0.13/0.00	0.13/0.00
GRE-E-F	–	–	–	0.48/0.00

[*] Each cell contains *x*/*y*, where *x* and *y* are A_{12} and *p*-value for pairwise comparison, respectively.

4.2.1 Answer to RQ2.1: Does the selection of coverage granularity level affect FL effectiveness?

The statistics in terms of *EXAM* score for different coverage granularity levels are shown in Table 4. Let us consider the TSR based on branch coverage information as an illustrative example. If we use branch coverage information together with 1 out of 3 TSR strategies and 1 out of 4 reduction metrics to perform TSR, there will be totally 3×4×1=12 TSR techniques. We adopted each of 12 TSR techniques to reduce the 226×1,000 OTS, performed each of 4 SBFL techniques on the RS, and then obtained 226×1,000×3×4×1×4 *EXAM* scores. Finally, we took the average across the 226×1,000×3×4×1×4 *EXAM* scores.

Because the *p*-value of the Kruskal-Wallis test is 0.00, we further performed the statistical tests for pairwise comparison. According to the *p*-values for all pairs, the difference between any two of the three coverage granularity levels is statistically significant. Furthermore, only the A_{12} values of the two pairs including function coverage level are less

Table 4. The statistics in *EXAM* score (%) and the pairwise comparisons for RQ 2.1.

Coverage granularity level	Mean	Median	*p*-value[1]	Pairwise coverage granularity levels	A_{12}	*p*-value[2]
branch	18.15	15.12	0.00	branch vs statement	0.38	0.00
function	39.01	40.00		branch vs function	0.16	0.00
statement	22.32	20.84		statement vs function	0.21	0.00

[1]It indicates the statistical result of the Kruskal-Wallis test. [2]It indicates the statistical result of the Mann-Whitney *U* test.

than 0.29, which is indicative of a "large" effect size. That is, performing TSR based on branch coverage information can achieve significantly but not considerably better FL effectiveness than that based on statement coverage information. Yet, the results show that the FL effectiveness achieved by performing TSR based on function coverage information is significantly and considerably worse than those based on the others.

According to the aforementioned descriptions, performing TSR based on branch coverage information should be a desirable choice for maximizing FL effectiveness. We also posit that the selection of coverage granularity level may lead to considerable impact on FL effectiveness.

4.2.2 Answer to RQ2.2: Does the selection of the reduction metric affect FL effectiveness?

To facilitate the explanation, let us consider the *EXAM* score achieved by the reduction metric Cov as an illustrative example. We utilized Cov together with 1 out of 3 TSR strategies and 1 out of 3 coverage granularity levels to reduce test suites, thus resulting in 3×1×3=9 different TSR techniques. We reduced the 226×1,000 OTS by means of each of 9 TSR techniques, adopted each of the 4 SBFL techniques to locate faults based on each of the 226×1,000×3×1×3 RS, and then obtained the 226×1,000×3×1×3×4 *EXAM* scores. Finally, we took the average across the 226×1,000×3×1×3×4 *EXAM* scores. The statistics in terms of *EXAM* score for different reduction metrics are shown in Table 5.

Table 5. The statistics in *EXAM* score (%) and the pairwise comparisons for RQ2.2.

Reduction metric	Mean	Median	*p*-value[1]	Pairwise reduction metrics	A_{12}	*p*-value[2]
Cov	27.21	25.01	0.00	Irre vs Cov	0.46	0.00
Ratio	26.68	24.36		Irre vs Ratio	0.47	0.00
Irre	25.21	22.78		Irre vs EIrre	0.46	0.00
EIrre	26.88	24.56		Ratio vs Cov	0.49	0.00
				Ratio vs EIrre	0.50	0.00
				EIrre vs Cov	0.49	0.00

[1]It indicates the statistical result of the Kruskal-Wallis test. [2]It indicates the statistical result of the Mann-Whitney *U* test.

According to Table 5, we found that there exists the difference among these 4 reduction metrics because the *p*-value of the Kruskal-Wallis test is less than 0.05. The results of the statistical tests for pairwise comparison further indicate that the FL effectiveness caused by Irre is significantly better than that of the others; on the other hand, Cov works significantly worse than any of the other reduction metrics (*i.e.*, A_{12}<0.50 and *p*-value=0.00 for the corresponding pairs). It is also noted that A_{12} between Irre and Cov is close to 0.50, thus indicating a "small" effect size. That is, the difference in FL effectiveness between the best and the worst reduction metrics is not considerable.

Therefore, it is better to adopt Irre as the reduction metric during TSR in order to maximize FL effectiveness. Yet, our experiment also indicate that reduction metric may not be a key factor to affect FL effectiveness.

4.2.3 Answer to RQ2.3: Does the selection of TSR strategy affect FL effectiveness?

Table 6 gives the statistics in terms of *EXAM* score for adopting different TSR strategies. For each *EXAM* score in this table, we adopted 1×4×3=12 TSR techniques (*i.e.*, a single TSR strategy together with 1 out of 4 reduction metrics and 1 out of 3 coverage granularity levels) to reduce the 226×1,000 OTS. We further performed each of 4 SBFL techniques based on each of the 226×1,000×1×4×3 RS, calculated the *EXAM* score for each RS, and took the average across these 226×1,000×1×4×3×4 *EXAM* scores.

As seen from Table 6, the *p*-value of Kruskal-Wallis test indicates that there is a significant difference in *EXAM* score among these 3 TSR strategies. Thus, we performed the statistical tests for pairwise comparison. As is clear from the last two columns, AG works significantly better than GRE and HGS because both of the A_{12} values and the *p*-values for these two pairs are less than the thresholds, while HGS performs significantly better than GRE. Therefore, our experiment shows that AG significantly outperforms the others from the standpoint of FL effectiveness. It is noted that all three A_{12} values are close to 0.50.

Therefore, the AG algorithm is recommended to perform TSR for retaining FL effectiveness. Yet, similar to the selection of reduction metric, the selection of TSR strategy may not be the most important factor to affect FL effectiveness.

Table 6. The statistics in *EXAM* score (%) and the pairwise comparisons for RQ2.3.

TSR strategy	Mean	Median	*p*-value[1]	Pairwise TSR strategies	A_{12}	*p*-value[2]
AG	25.61	23.51	0.00	AG vs HGS	0.48	0.00
GRE	27.34	24.75		AG vs GRE	0.47	0.00
HGS	26.53	23.93		HGS vs GRE	0.48	0.00

[1]It indicates the statistical result of the Kruskal-Wallis test. [2]It indicates the statistical result of the Mann-Whitney *U* test.

4.2.4 Answer to RQ2.4: Does the selection of SBFL technique affect FL effectiveness?

Similar to the replies of RQs 2.1 through 2.3, we performed each of 4 SBFL techniques on 226×1,000 OTS and 226×1,000×3×4×3 RS, respectively, obtained the (226×1,000)+(226×1,000×3×4×3) *EXAM* scores, and then took the average across these *EXAM* scores. Table 7 exhibits the statistics in terms of *EXAM* score for different SBFL techniques. Because the *p*-value of Kruskal-Wallis test is 0.00, we performed the statistical tests for pairwise comparison. The *p*-values of the Mann-Whitney U test show that Jaccard performs the best and Tarantula the worst. Additionally, the values of A_{12} for all 6 pairs are far from 0.50, thus indicating that the difference between any two of these techniques is almost considerable. As such, in comparison to the other 3 SBFL techniques, Jaccard is recommended for including in the CI.

4.3 Answer to RQ3: What is the most recommended CI configuration for optimizing FL effectiveness?

In our study, the number of all CI configurations combining the 37 TSR techniques (including *OTS*) and 4 SBFL techniques is 37×4=148 and showing all of them will take a

Table 7. The statistics in *EXAM* score (%) and the pairwise comparisons for RQ 2.4.

SBFL technique	Mean	Median	p-value[1]	Pairwise SBFL techniques	A_{12}	p-value[2]
Jaccard	26.01	22.54	0.00	Jaccard vs Ochiai	0.29	0.00
Ochiai	30.95	27.37		Jaccard vs SBI	0.22	0.00
SBI	36.22	32.51		Jaccard vs Tarantula	0.19	0.00
Tarantula	41.38	37.55		Ochiai vs SBI	0.28	0.00
				Ochiai vs Tarantula	0.22	0.00
				SBI vs Tarantula	0.28	0.00

[1]It indicates the statistical result of the Kruskal-Wallis test. [2]It indicates the statistical result of the Mann-Whitney *U* test.

Table 8. The statistics in *EXAM* score (%) and the pairwise comparisons for RQ3.

Configuration	Mean	Median	p-value[1]	Configuration pair	A_{12}	p-value[2]
OTS-J	13.04	6.39	0.00	*OTS-J* vs *AG-I-B-J*	0.27	0.00
AG-I-B-J	13.59	8.92		*OTS-J* vs *AG-I-S-J*	0.01	0.00
AG-I-S-J	16.01	12.16		*AG-I-B-J* vs *AG-I-S-J*	0.04	0.00

[1]It indicates the statistical result of the Kruskal-Wallis test. [2]It indicates the statistical result of the Mann-Whitney *U* test.

lot of space. Thus, Table 8 only includes the best 3 CI configurations in terms of *EXAM* score and shows their statistics. As seen from this table, the difference in *EXAM* score among these 3 configurations is statistically significant (the Kruskal-Wallis test's p-value < 0.05). Hence, the statistical tests for pairwise comparison were further performed. The A_{12} values and the p-values for all 3 pairs indicate that the differences between any two of the best 3 CI configurations are not only statistically significantly different but also considerable. Thus, we suggest *OTS-J* in order to maximize the FL effectiveness in a CI. In addition, this finding can be justified by the findings from RQ1 and RQ2.4 (*i.e.* the *OTS* and Jaccard are recommend for TSR and SBFL, respectively, to optimize the FL effectiveness in a CI).

4.4 Summary of the Experiment Results

We summarized our findings from the experimental results as follows: (1) The FL effectiveness achieved by the OTS is significantly better than that achieved by the RS that is produced by any of the 36 TSR techniques; (2) If software developers decide to adopt TSR in the CI, our experiment results suggest reducing test suites using *AG-I-B*. Yet, please notice that although the FL effectiveness of *AG-I-B* is close to that of *OTS*, their difference is statistically significant; (3) The selection of coverage granularity level may lead to more considerable impact on FL effectiveness than the other two TSR factors; (4) The SBFL technique, Jaccard, is recommended to locate faults in terms of *EXAM* score; (5) Finally, we evaluated the 148 configurations and found that the CI configuration *OTS-J* works significantly better than the others from the standpoint of FL effectiveness.

On the whole, the replies to RQs 2.1 through 2.3 confirm our finding in the reply to RQ1. More specifically, RQs 2.1 through 2.3 recommend the branch coverage information, the Irre metric, and the AG algorithm while RQ1 suggests that *AG-I-B* achieves the best SBFL effectiveness among all TSR techniques except for *OTS*.

4.5 Threats to Validity

Threats to internal validity: The implementation of TSR and SBFL techniques is our primary threat to internal validity. To mitigate this threat, we manually inspected our im-

plementation with small examples to check whether the results from both are consistent. Additionally, the selection of the criterion used to evaluate FL effectiveness (*i.e.*, the *EXAM* score) is another threat to internal validity. The *EXAM* score is a criterion that was frequently adopted to evaluate FL effectiveness. We adopted this criterion in order to make our experiment results to be comparable to prior work.

Threats to external validity: In this experiment, we utilized the 9 subjects (*i.e.*, the 7 programs in the Siemens suite, flex, and gzip) because they were commonly used in the relevant studies. Moreover, our experiment only focuses on single-fault versions. Hence, we cannot confirm that our findings will generally hold for the other subjects or subjects with multiple faults.

5. RELATED STUDIES

In the literature, Jiang *et al.* [15] discussed how the FL is impacted by different factors in test case prioritization (TCP) (*i.e.*, the strategy, the coverage granularity, and the time cost) during a CI process. Their empirical results found that the strategy and time cost of TCP techniques are key factors which influence FL effectiveness. They also found that, when sufficiently failed test cases are considered, FL effectiveness can be effective. Although Jiang *et al.*'s work is relevant to our study, their work focuses on the relationship between "TCP" and "FL". Yet, instead of "TCP", ours focuses on the relationship between "TSR" and "FL". Additionally, our work found that coverage granularity level for TSR has more significant impact on FL effectiveness than the other factors whereas Jiang *et al.*'s work claimed that coverage granularity level for TCP should not be the main factor.

To the best of our knowledge, few studies in the literature comprehensively discussed the relationship between "TSR" and "FL". Although Yu *et al.* [10] studied the impact of TSR on SBFL, the TSR strategy considered in their study only includes the Greedy algorithm. Additionally, Yu *et al.* focused their work on proposing a new TSR metric, called vector-based, to produce an RS that can lead to better FL effectiveness. Their results recommended that performing Greedy based on statement coverage information is appropriate if software developers' main concern is the testing cost. On the other hand, the statement-based metric should be replaced by vector-based information if their main consideration is FL effectiveness. Yet, Yu *et al.*'s study should not be sufficiently to help software developers understand whether reducing test suites using the other algorithms (*e.g.*, GRE and HGS) at the different coverage granularity levels (*i.e.*, branch and function) will impact the FL effectiveness. Additionally, because their work only evaluated the importance of test cases according the code coverage, we still cannot understand the impact caused by adopting another way to evaluate test cases during TSR.

The aforementioned differences confirm that our work is worthwhile to carry out even if the prior studies already have some findings. Thus, the focused studies of our paper and Jiang *et al.*'s and Yu *et al.*'s work are different and may be complementary of each other.

6. CONCLUSIONS

TSR and FL are two approaches to improve a CI process. Yet, there exist several factors to control TSR and FL, thus complicating the configuration of a CI. This paper aims

to discuss the impact on FL effectiveness caused by the 4 factors for CI configurations (including the 3 TSR factors and the selection of SBFL technique). Our empirical results indicate that performing regression testing based on the OTS is more recommendable than the RS from the standpoint of optimizing FL effectiveness. If TSR will still be included in the CI, the selection of coverage granularity level, especially branch coverage information, leads to more impact on FL effectiveness than the other two TSR factors. Additionally, our experiment also confirms that the selection of SBFL technique is important for maximizing FL effectiveness. More specifically, among the 4 SBFL techniques under study, Jaccard is the most recommendable. Overall, if software developers' main consideration is increasing the FL effectiveness, we suggest using the OTS to perform regression testing and adopting the Jaccard to pinpoint the location of revealed faults.

This paper aims to develop a guidance for software developers to design a CI environment where TSR and SBFL are considered. To the best of our knowledge, this is the first work to carry out such an interesting and important task. In future work, we plan to cover more TSR and FL techniques and perform additional experiments on more large-scale real programs with multiple faults. For example, this paper only focuses our experiment on SBFL. It may be worthwhile to understand the effect of applying the other types of FL techniques (*e.g.*, slice-based, machine learning-based, data mining-based, model-based [12]) in a CI. We also plan to investigate whether our findings hold for the subjects of different scales. Additionally, we plan to conduct the experiments on the multiple-fault programs. These manners will decrease the threats to validity that are described in Section 4.5 and make our experiment approach to reality. On the whole, the combination of this paper's suggestion and the achievements completed during future work may provide a more valuable and complete guidance for software developers to design their CI.

REFERENCES

1. H. K. N. Leung and L. White, "Insights into regression testing," in *Proceedings of International Conference on Software Maintenance*, 1989, pp. 60-69.
2. J. M. Kim and A. Porter, "A history-based test prioritization technique for regression testing in resource constrained environments," in *Proceedings of the 24th International Conference on Software Engineering*, 2002, pp. 119-129.
3. M. Pezzè and M. Young, *Software Testing and Analysis: Process, Principles, and Techniques*, John Wiley & Sons, NY, 2008.
4. G. Rothermel, M. J. Harrold, J. Von Ronne, and C. Hong, "Empirical studies of test-suite reduction," *Software Testing, Verification and Reliability*, Vol. 12, 2002, pp. 219-249.
5. S. Parsa and A. Khalilian, "A bi-objective model inspired greedy algorithm for test suite minimization," in *Proceedings of International Conference on Future Generation Information Technology*, 2009, pp. 208-215.
6. S. Tallam and N. Gupta, "A concept analysis inspired greedy algorithm for test suite minimization," in *Proceedings of the 6th ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering*, 2005, pp. 35-42.
7. D. Jeffrey and N. Gupta, "Improving fault detection capability by selectively retaining test cases during test suite reduction," *IEEE Transactions on Software Engineering*,

Vol. 33, 2007, pp. 108-123.
8. D. Gong, T. Wang, X. Su, and P. Ma, "A test-suite reduction approach to improving fault-localization effectiveness," *Computer Languages, Systems & Structures*, Vol. 39, 2013, pp. 95-108.
9. W. Fu, H. Yu, G. Fan, X. Ji, and X. Pei, "A test suite reduction approach to improving the effectiveness of fault localization," in *Proceedings of International Conference on Software Analysis, Testing and Evolution*, 2017, pp. 10-19.
10. Y. Yu, J. Jones, and 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*, 2008, pp. 201-210.
11. W. E. Wong, V. Debroy, R. Gao, and Y. Li, "The DStar method for effective software fault localization," *IEEE Transactions on Reliability*, Vol. 63, 2013, pp. 290-308.
12. W. E. Wong, R. Gao, Y. Li, R. Abreu, and F. Wotawa, "A survey on software fault localization," *IEEE Transactions on Software Engineering*, Vol. 42, 2016, pp. 707-740.
13. H. A. de Souza, M. L. Chaim, and F. Kon, "Spectrum-based software fault localization: A survey of techniques, advances, and challenges," *arXiv Preprint*, 2016, arXiv:1607.04347v2.
14. P. M. Duvall, S. Matyas, and A. Glover, *Continuous Integration: Improving Software Quality and Reducing Risk*, Pearson Education, NY, 2007.
15. B. Jiang, Z. Zhang, W. K. Chan, T. H. Tse, and T. Y. Chen, "How well does test case prioritization integrate with statistical fault localization?" *Information and Software Technology*, Vol. 54, 2012, pp. 739-758.
16. K. C. Wang, T. T. Wang, and X. H. Su, "Test case selection using multi-criteria optimization for effective fault localization," *Computing*, Vol. 100, 2018, pp. 787-808.
17. J. Intasara, C. T. Lin, and A. Srisawat, "Evaluating the influence on the effectiveness of software fault localization caused by the regression test suite reduction techniques in continuous integration process," in *Proceedings of the 11th International Conference on Advances in Databases, Knowledge, and Data Applications*, 2019.
18. S. Yoo and M. Harman, "Regression testing minimization, selection and prioritization: a survey," *Software Testing, Verification and Reliability*, Vol. 22, 2012, pp. 67-120.
19. A. Shi, T. Yung, A. Gyori, and D. Marinov, "Comparing and combining test-suite reduction and regression test selection," in *Proceedings of the 10th Joint Meeting on Foundations of Software Engineering*, 2015, pp. 237-247.
20. C. T. Lin, K. W. Tang, and G. M. Kapfhammer, "Test suite reduction methods that decrease regression testing costs by identifying irreplaceable tests," *Information and Software Technology*, Vol. 56, 2014, pp. 1322-1344.
21. L. Zhang, D. Marinov, and L. Zhang, "An empirical study of junit test-suite reduction," in *Proceedings of the 22nd International Symposium on Software Reliability Engineering*, 2011, pp. 170-179.
22. D. Jeffrey and N. Gupta, "Test suite reduction with selective redundancy," in *Proceedings of the 21st International Conference on Software Maintenance*, 2005, pp. 549-558.
23. J. W. Lin and C. Y. Huang, "Analysis of test suite reduction with enhanced tie-breaking techniques," *Information and Software Technology*, Vol. 51, 2009, pp. 679-690.
24. P. C. Jorgensen, *Software Testing: A Craftsman's Approach*, CRC Press, FL, 2007.

25. H. Zhong, L. Zhang, and H. Mei, "An experimental study of four typical test suite reduction techniques," *Information and Software Technology*, Vol. 50, 2008, pp. 534-546.
26. X. Y. Ma, Z. F. He, B. K. Sheng, and C. Q. Ye, "A genetic algorithm for test-suite reduction," in *Proceedings of IEEE International Conference on Systems, Man and Cybernetics*, 2005, Vol. 1, pp. 133-139.
27. A. M. Smith and G. M. Kapfhammer, "An empirical study of incorporating cost into test suite reduction and prioritization," in *Proceedings of ACM Symposium on Applied Computing*, 2009, pp. 461-467.
28. C. T. Lin, K. W. Tang, J. S. Wang, and G. M. Kapfhammer, "Empirically evaluating Greedy-based test suite reduction methods at different levels of test suite complexity," *Science of Computer Programming*, Vol. 150, 2017, pp. 1-25.
29. T. Y. Chen and M. F. Lau, "A new heuristic for test suite reduction," *Information and Software Technology*, Vol. 40, 1998, pp. 347-354.
30. M. J. Harrold, R. Gupta, and M. L. Soffa, "A methodology for controlling the size of a test suite," *ACM Transactions on Software Engineering and Methodology*, Vol. 2, 1993, pp. 270-285.
31. J. A. Jones and M. J. Harrold, "Empirical evaluation of the tarantula automatic fault-localization technique," in *Proceedings of the 20th International Conference on Automated Software Engineering*, 2005, pp. 273-282.
32. R. Abreu, P. Zoeteweij, R. Golsteijn, and A. J. Van Gemund, "A practical evaluation of spectrum-based fault localization," *Journal of Systems and Software*, Vol. 82, 2009, pp. 1780-1792.
33. B. Liblit, M. Naik, A. X. Zheng, A. Aiken, and M. I. Jordan, "Scalable statistical bug isolation," in *Proceedings of ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2005, pp. 15-26.
34. W. E. Wong, V. Debroy, and B. Choi, "A family of code coverage-based heuristics for effective fault localization," *Journal of Systems and Software*, Vol. 83, 2010, pp. 188-208.
35. W. E. Wong, V. Debroy, and D. Xu, "Towards better fault localization: A crosstab-based statistical approach," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, Vol. 42, 2011, pp. 378-396.
36. M. Renieres and S. P. Reiss, "Fault localization with nearest neighbor queries," in *Proceedings of the 18th International Conference on Automated Software Engineering*, 2003, pp. 30-39.
37. A. Ochiai, "Zoogeographic studies on the soleoid fishes found in Japan and its neighbouring regions," *Bulletin of the Japanese Society of Scientific Fisheries*, Vol. 22, 1957, pp. 526-530.
38. A. Meyer, A. Garcia, and A. Souza, "Comparison of similarity coefficients used for cluster analysis with dominant markers in maize (Zea mays L)," *Genetics and Molecular Biology*, Vol. 27, 2004, pp. 83-91.
39. K. Lage, N. T. Hansen, E. O. Karlberg, A. C. Eklund, F. S. Roque, P. K. Donahoe, Z. Szallasi, T. S. Jensen, and S. Brunak, "A large-scale analysis of tissue-specific pathology and gene expression of human disease genes and complexes," in *Proceedings of the National Academy of Sciences*, Vol. 105, 2008, pp. 20870-20875.
40. L. Naish, H. J. Lee, and K. Ramamohanarao, "A model for spectra-based software

diagnosis," *ACM Transactions on Software Engineering Methodology*, Vol. 20, 2011, pp. 1-32.
41. B. Baudry, F. Fleurey, and Y. Le Traon, "Improving test suites for efficient fault localization," in *Proceedings of the 28th International Conference on Software Engineering*, 2006, pp. 82-91.
42. C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in Software Engineering*, Springer Science & Business Media, Berlin, 2012.
43. A. Vargha and H. D. Delaney, "A critique and improvement of the CL common language effect size statistics of McGraw and Wong," *Educational and Behavioral Statistics*, Vol. 25, 2000, pp.101-132.
44. H. Do, S. Elbaum, and G. Rothermel, "Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact," *Empirical Software Engineering*, Vol. 10, 2005, pp. 405-435.

Jutarporn Intasara (龐之茵) received the B.S. and M.S. degrees in Computer Science from Prince of Songkla University, Songkhla, Thailand, in 2008 and 2011, respectively. She is currently pursuing the Ph.D. degree in the Computer Science and Information Engineering, National Chiayi University. Her current research interests include software testing and debugging.

Chu-Ti Lin (林楚迪) received the B.S. and Ph.D. degrees in Computer Science from the National Tsing Hua University, Hsinchu, Taiwan, in 2003 and 2009, respectively. He is currently an Associate Professor in the Department of Computer Science and Information Engineering at National Chiayi University, Chiayi, Taiwan. He has authored or coauthored research results in venues such as IEEE Transactions on Software Engineering, IEEE Transactions on Computers, IEEE Transactions on Reliability, Information and Software Technology, Journal of Systems and Software, Science of Computer Programming. His research interests include software engineering, software reliability engineering, software testing, software debugging, and software testability.