Information and Software Technology 51 (2009) 1573–1585

Contents lists available at ScienceDirect

Information and Software Technology

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

Is non-parametric hypothesis testing model robust for statistical fault localization? ☆,☆☆

Zhenyu Zhang [a], W.K. Chan [b,*], T.H. Tse [a], Peifeng Hu [c], Xinming Wang [d]

[a] *Department of Computer Science, The University of Hong Kong, Pokfulam, Hong Kong*
[b] *Department of Computer Science, City University of Hong Kong, Tat Chee Avenue, Hong Kong*
[c] *China Merchants Bank, Central, Hong Kong*
[d] *Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Clear Water Bay, Hong Kong*

ARTICLE INFO

Article history:
Available online 8 July 2009

Keywords:
Fault localization
Non-parametric
Hypothesis testing
Normality

ABSTRACT

Fault localization is one of the most difficult activities in software debugging. Many existing statistical fault-localization techniques estimate the fault positions of programs by comparing the program feature spectra between passed runs and failed runs. Some existing approaches develop estimation formulas based on mean values of the underlying program feature spectra and their distributions alike. Our previous work advocates the use of a non-parametric approach in estimation formulas to pinpoint fault-relevant positions. It is worthy of further study to resolve the two schools of thought by examining the fundamental, underlying properties of distributions related to fault localization. In particular, we ask: Can the feature spectra of program elements be safely considered as normal distributions so that parametric techniques can be soundly and powerfully applied? In this paper, we empirically investigate this question from the program predicate perspective. We conduct an experimental study based on the Siemens suite of programs. We examine the degree of normality on the distributions of evaluation biases of the predicates, and obtain three major results from the study. First, almost all examined distributions of evaluation biases are either normal or far from normal, but not in between. Second, the most fault-relevant predicates are less likely to exhibit normal distributions in terms of evaluation biases than other predicates. Our results show that normality is not common as far as evaluation bias can represent. Furthermore, the effectiveness of our non-parametric predicate-based fault-localization technique weakly correlates with the distributions of evaluation biases, making the technique robust to this type of uncertainty in the underlying program spectra.

1. Introduction

Software debugging is time-consuming and is often a bottleneck in software development process. It involves at least two crucial steps, namely *fault localization* and *fault correction*. Fault localization identifies the causes of abnormal behaviors of a faulty program. Fault correction modifies the faulty program or data structure to eliminate the effect of the identified faults.

A traditional fault-localization technique consists of setting breakpoints, re-executing the faulty program on the inputs, and examining the corresponding program states [13]. Recently, statistical fault-localization techniques [10,12,14–17] were proposed and reported to be promising. They locate faults by analyzing the statistics of dynamic program behaviors. A *failed run* is a program execution that reveals a failure, and a *passed run* is a program execution that reveals no failure. A statistical fault-localization technique locates a fault-relevant statement (or a faulty statement directly) by comparing the statistical information of program elements in these two kinds of runs. Such program elements can be statements [12] or predicates [14,15].

Because of their statistical nature, these techniques assume that there are statistically enough passed runs and failed runs to locate faults collectively. These techniques build underlying statistical behavior models for the aggregated execution data of selected program elements (which we call *features*), and search for program elements that strongly correlate with the observed program failures. For instance, predicate-based statistical techniques [14–17] locate those program predicates strongly related to faults. A *program predicate* is a Boolean expression about the property of a

☆ This research is supported in part by GRF grants of the Research Grants Council of Hong Kong (Project Nos. 123207 and 716507) and SRG grants of City University of Hong Kong (Project Nos. 7002324 and 7002464).

☆☆ A preliminary version of this paper was presented at the 8th International Conference on Quality Software (QSIC 2008) [8].

* Corresponding author. Tel.: +852 2788 9684; fax: +852 2788 8614.
E-mail addresses: zyzhang@cs.hku.hk (Z. Zhang), wkchan@cs.cityu.edu.hk (W.K. Chan), thtse@cs.hku.hk (T.H. Tse), pfhu@cs.hku.hk (P. Hu), rubin@cse.ust.hk (X. Wang).

0950-5849/$ - see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.infsof.2009.06.013

system at some program location (such as a statement). CBI [14,15] checks the probability of a predicate to be evaluated to be true in all failed runs as well as the probability in all the runs (irrespectively of whether passed or failed), and measures the *increase* from the former to the latter. CBI uses this increase as a ranking score, which indicates how much the predicate is related to a fault. SOBER [16,17] defines *evaluation bias* to model the chance that a predicate is evaluated to be true in each run. More precisely, if P is a predicate and $\pi(P)$ is the probability that it is evaluated to be true in every run, then $\pi(P)$ is estimated to be $\frac{n_t}{n_t+n_f}$, where n_t is the number of times that P is evaluated to be true and n_f is the number of times that P is evaluated to be false. SOBER then evaluates the difference between the distributions of $\pi(P)$ between passed runs and failed runs, and deems that the larger the difference, the more will P be relevant to a fault.

As indicated in their models, CBI uses means and changes in mean values to estimate the fault relevance of a program predicate; SOBER applies the central limit theorem in statistics to measure the behavioral difference of a predicate between passed runs and failed runs. Typically, a mean value may reasonably represent a distribution if the variable of the distribution tends to cluster around the mean value. *Is it suitable to assume any known distribution in the program behaviors such as the evaluation biases of predicates?* We have conducted an initial study in our previous work [8] and found that evaluation biases may not form normal distributions. Our previous work also proposes to use a standard non-parametric hypothesis testing method to compare the program spectra of passed runs and those of failed runs. We have stipulated our model in the context of predicate-based statistical fault localization, and picked a form of the Mann–Whitney test to determine the degree of difference between the evaluation biases for passed runs and those for failed runs. The degree of difference in such a comparison is used as the ranking score, which indicates how much a predicate is related to a fault. Based on the ranking scores of the predicates, we reorder the predicates accordingly (predicates having higher values in ranking score are deemed to be more suspicious). The empirical results [8] on the Siemens suite show that our technique can be effective and outperforms CBI and SOBER in locating faults.

In view of the above-mentioned initial study, in this paper, we extend our investigation and ask a dual-sided question: Can the feature spectra of program elements be safely considered as normal distributions so that parametric fault-localization techniques can be soundly and powerfully applied? Alternatively, to what extent can such program spectra be regarded as normal distributions? If the answers to these questions are negative, we further ask the following question: Can the effectiveness of non-parametric fault-localization techniques be really decoupled from the distribution shape of the program spectra?

In this paper, we collect the evaluation biases of all the predicates from passed runs and those from failed runs, and conduct normality tests on them. By using standard statistical hypothesis testing, we successfully reject the assumption that normal distribution is commonly exhibited by evaluation biases of predicates. We further investigate the effect of such normality property for predicates on fault-localization techniques. The empirical results show that the effectiveness of our proposal for non-parametric fault localization [8] weakly correlates with the presumed normal distribution of evaluation biases.

The main contribution of the paper is fourfold: (i) It is the first investigation on the normality nature of the execution spectra. The empirical results show that normal distribution is not common for the evaluation biases of predicates. In particular, the results indicate that the chance of the distribution of the evaluation biases of fault-relevant predicates being normal is less likely than that of other predicates. (ii) Such a finding highlights a threat to the construct validity of any empirical study which is based on the assumption that the evaluation biases of predicates form normal distributions. (iii) It proposes a new metric *P*-score to measure the effectiveness of fault-localization techniques. (iv) It investigates the effect of normality for the evaluation biases of predicates on non-parametric fault-localization techniques. The empirical results show that the effectiveness of our non-parametric fault-localization technique weakly correlates with the normality of the underlying distribution of evaluation biases.

The remainder of the paper is organized as follows. Section 2 gives a motivating study. Section 3 revisits the background and sets the scene for the empirical study. Research questions are outlined in Section 4, followed by the experiment in Section 5. A literature review is given in Section 6. Finally, Section 7 concludes the paper.

2. Motivating study

In this section, we use one of the Siemens programs [5] to illustrate our important initial finding on the statistics of program behaviors. Fig. 1 shows the code excerpted from faulty version “v1” of the program “tot_info”. In this code fragment, seven predicates are included, labeled as P_1 to P_7. The statement “goto ret1;” (labeled as E_1) is intentionally commented out by the Siemens researchers to simulate a statement omission fault. Locating such a kind of fault is often difficult even if the execution of a failed test case is traced step-by-step.

```
P1:     if ( rdf ≤ 0 || cdf ≤ 0 ) {
            info = -3.0;
            goto ret3;
        }
        :
P2:     for ( i = 0; i < r; ++i ) {
            double sum = 0.0;
P3:         for ( j = 0; j < c; ++j ) {
                long k = x(i,j);
P4:             if ( k < 0L ){
                    info = -2.0;
E1:                 /*goto ret1;*/
                }
                sum += (double)k;
            }
            N + = xi[i] = sum;
        }
P5:     if ( N ≤ 0.0 ) {
            info = -1.0;
            goto ret1;
        }
P6:     for ( j = 0; j < c; ++j ) {
            double sum = 0.0;
P7:         for ( i = 0; i < r; ++i )
                sum += (double)x(i,j);
            xj[j] = sum;
        }
        :
        ret1:
```

Fig. 1. Excerpt from faulty version “v1” of program “tot_info” from the Siemens programs.

Let us focus on program behaviors resulting from predicate evaluations because they have been successfully used in previous fault-localization research such as SOBER. We observe that the predicate "$P_4:$ k < 0L" is highly relevant to program failures because the omitted statement E_1 is in the *true* block of the branch structure of statement P_4. We further find that none of the predicates P_1, P_2, and P_3 is related to failures because they neither directly activate the fault nor propagate an error. Predicate P_5 is also related to the fault, since commenting out the goto statement (E_1) will render a higher chance for P_5 to be evaluated. Predicates P_6 and P_7 are increasingly distant from the faulty statement E_1.

The seven pairs of distributions of evaluation biases with respect to P_1 to P_7 are shown via the histograms in Fig. 2. To attain clear explanations, we also zoom into the histograms for P_7 in Fig. 3.

In each of these plots, the *X*-axis stands for the evaluation biases (varying in the range of [0, 1]), and the *Y*-axis is the number of (passed or failed) runs that share the same value of evaluation bias. They are produced by executing the program over all the test cases in the Siemens suite. If a predicate is not executed in a run, there will be no data captured in the distribution.

The series of histograms (distributions of evaluation biases) on the left are for the passed runs and those on the right are for the failed runs. The resolution (step) of the histograms is 0.01. Take the plot in Fig. 3a as an illustration. The leftmost bar means that there are 82 successful test cases, over which the evaluation biases of P_7 in their corresponding program execution are in the range of [0.65, 0.66).

In previous work such as SOBER [16,17], the statistical parameters *mean* and *variance* are used to estimate the similarity between such a pair of distributions of evaluation biases. Let us take the pair of distributions with respect to P_1 as an illustration. Using the sample of evaluation biases for passed runs, the mean and variance of evaluation biases for passed runs are $\mu_p = 0.111$ and $\sigma_p = 0.390$, respectively. SOBER constructs the null hypothesis as "$\mu_p = \mu_f$ and $\sigma_p = \sigma_f$" [16,17], where $\mu_f = 0.073$ and $\sigma_f = 0.153$ stand for the mean and variance, respectively, of evaluation biases for failed runs. Following this hypothesis and the central limit theorem [29], μ_f should conform to $N\left(\mu_p, \frac{\sigma_p^2}{s}\right) = N(0.111, 0.001)$, where $s = 130$ is the sample size of evaluation biases for failed runs. Next, the similarity between the pair of distributions is measured by the likelihood function (μ_p, σ_p) when μ_f is given [16,17]. Hence, the likelihood function $L(\mu_p, \sigma_p|\mu_f) = \frac{\sqrt{s}}{\sigma_p}\varphi\left(\frac{\mu_f-\mu_p}{\sigma_p/\sqrt{s}}\right) = 0.399$, where φ is the probability density function of $N(0,1)$ [29]. The smaller the value, the less similar will be the pair of distributions.

We have the following observations from the histograms:

- *O*1: The two histograms for predicate P_1 resemble each other. The same phenomenon is observed for P_2, P_3, P_6, and P_7. Those for P_4 and P_5, however, differ significantly. Since predicates P_4 and P_5 are more fault relevant, it indicates that the differences of distributions between passed runs and failed runs can be good indicators of the fault relevance of predicates.
- *O*2: None of the histograms in Figs. 2 and 3 resembles a normal distribution. For each predicate of this program, we have conducted the standard Jarque–Bera test to determine whether its evaluation bias follows a normal distribution. The results show that, as far as the programs under study can represent, it is unrealistic to assume normal distributions for the evaluation biases of predicates.

From the above observations, the assumption that the evaluation biases of predicates form normal distributions may not be well supported by the empirical data. Furthermore, only a small number of test cases can reveal failures in practice, and the number of successful test cases is not large either. Because of that, in our previous work [8], we proposed a non-parametric hypothesis testing model, and advocated the use of a non-parametric predicate-based fault-localization technique. Is this program an exception? Can the feature spectra of program entities be safely considered as normal distributions so that parametric techniques can be applied rigorously? If the answer is negative, it is natural to develop and apply non-parametric techniques to support statistical fault localization. This motivates the study in this paper.

3. Background

In this section, we revisit the concept of evaluation bias and our non-parametric fault-localization technique.

3.1. Evaluation bias

Liblit et al. [14,15] propose three types of program location, namely *branches*, *returns*, and *scalar-pairs*, to sample the execution statistics of passed and failed runs. Each program location is associated with a set of Boolean predicates (known as program predicates). Each program predicate may be executed more than once in a run. Each evaluation will give either a true or a false value. It thus gives the notion of evaluation bias to estimate the probability of a predicate being evaluated as true in a run as follows:

Definition 1 (*Evaluation bias [17]*). Let n_t be the number of times that a predicate *P* has been evaluated to be true in a run, and n_f the number of times that it has been evaluated to be false in the same run. $\pi(P) = \frac{n_t}{n_t+n_f}$ is called **evaluation bias** of predicate *P* in this particular run.

3.2. Problem formulation

Let us consider a faulty program with a number of predicates. The set of predicates is denoted by $\{P_1, P_2, \ldots, P_i, \ldots, P_m\}$. We further let *R* and *R′* be the sets of passed runs and failed runs, respectively. For each passed run $r_j \in R$, the evaluation bias of predicate p_i is denoted by $E_{i,j}$. Similarly, for each failed run $r'_k \in R'$, the evaluation bias of predicate P_i is denoted by $E'_{i,k}$. The goal is to generate a predicate list, which is a permutation of $\{P_1, P_2, \ldots, P_i, \ldots, P_m\}$, such that the predicates are sorted in descending order of how much each P_i is fault relevant.

3.3. Non-parametric hypothesis testing model and non-parametric fault-localization technique

Following the convention in standard statistics, the non-parametric hypothesis testing model in our previous work [8] treats each run as an independent event. Let *H* and *H′* be the sets of possible passed runs and failed runs, respectively. Given a random run $r \in H$ or $r' \in H'$, let X_i be the random variable representing the evaluation bias of predicate P_i for the program execution over *r* or *r′*. Let us use $f(X_i|\theta_H)$ and $f(X_i|\theta_{H'})$ to denote the probability density functions of the evaluation biases of predicate P_i on *H* and *H′*, respectively. Suppose P_i correlates with the failures. Ideally, it should have a strong positive correlation with $\theta_{H'}$ and a strong negative correlation with θ_H. Similarly, suppose there is another predicate P_j that weakly correlates with the observed failures by the test cases. Ideally, P_j should have a weak (or no) correlation with $\theta_{H'}$ and a weak (or no) negative correlation with θ_H. Observe that the value difference in terms of the correlation values between (θ_H and $f(X_i|\theta_H)$) and ($\theta_{H'}$ and $f(X_i|\theta_{H'})$) forms a scalar metric. The

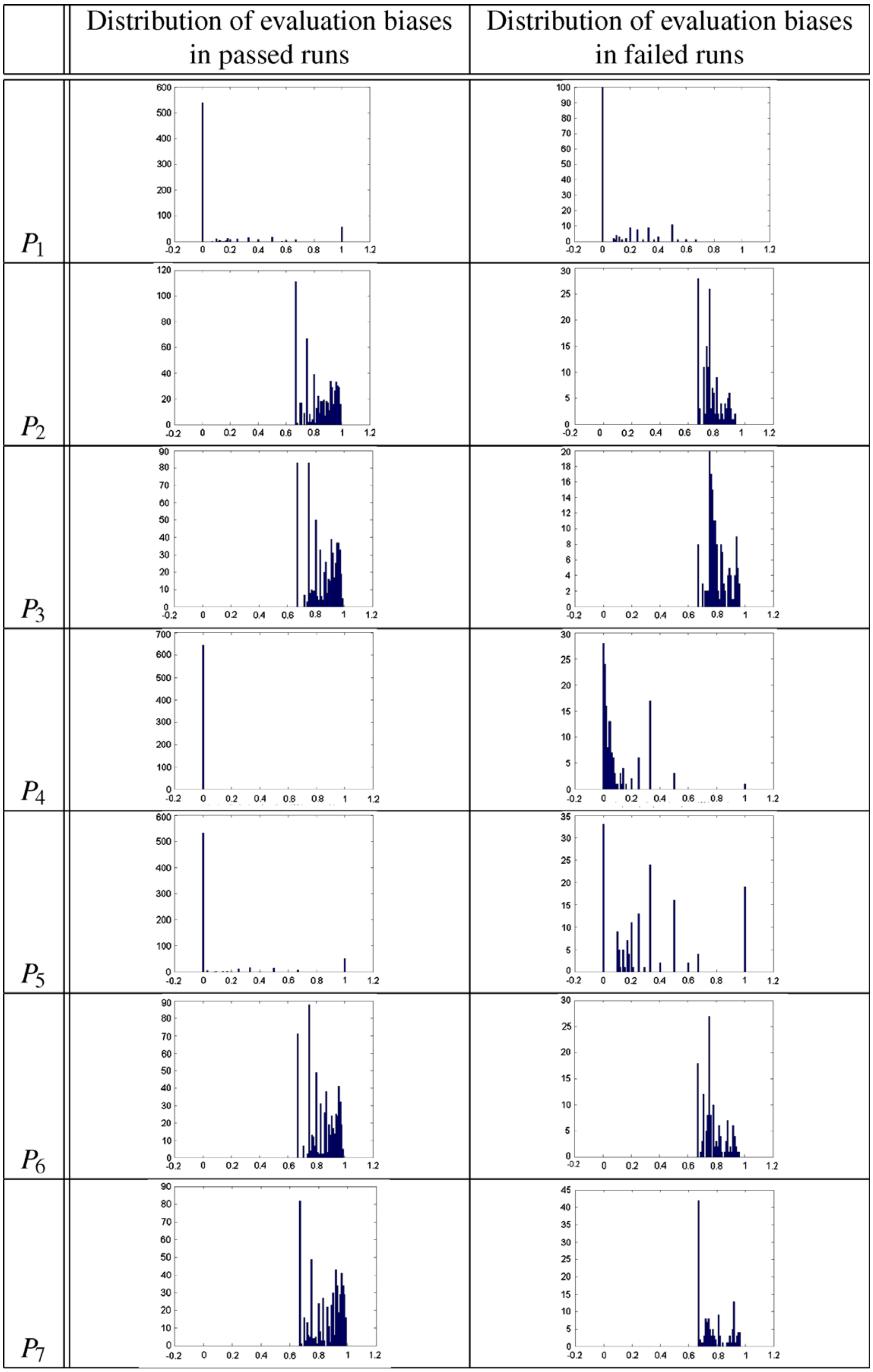


Fig. 2. Distributions of evaluation biases of predicates P_1 to P_7.

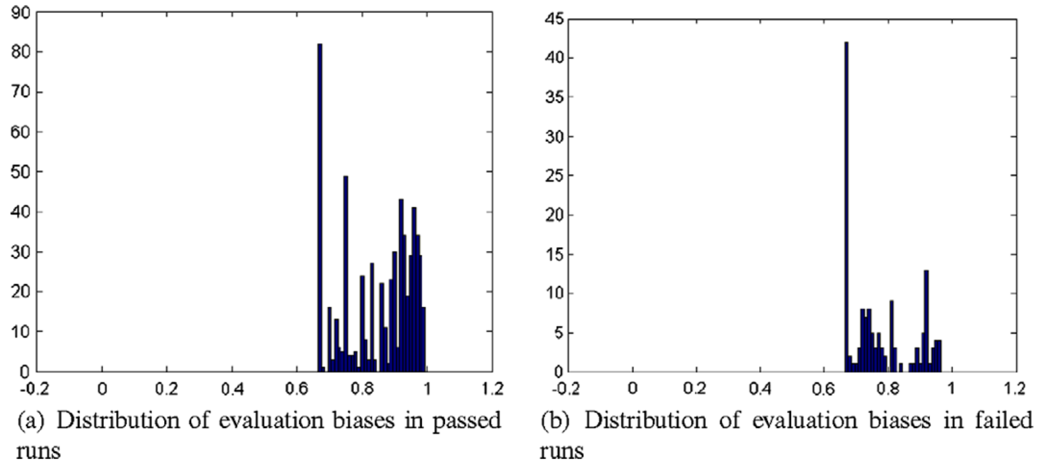


Fig. 3. Distributions of evaluation biases of predicate P_7.

larger the metric value, the better will be the chance that the corresponding predicate correlates with the observed failures. Identifying the predicates receiving (relatively) high values in terms of this metric would be important in statistical fault localization.

Our non-parametric hypothesis testing model then defines a ranking function

$$R(P_i) = Diff(f(X_i|\theta_H), f(X_i|\theta_{H'})) \tag{1}$$

to measure the difference between $f(X_i|\theta_H)$ and $f(X_i|\theta_{H'})$. Without any prior knowledge of $f(X_i|\theta_H)$ or $f(X_i|\theta_{H'})$, the model estimates them from the sample set, such as a test suite associated to the program. The evaluation bias of predicate P_i in the corresponding run of a test case from the test suite is treated as a sample of the random variable X_i. In this way, it obtains sample sets for $f(X_i|\theta_H)$ and $f(X_i|\theta_{H'})$, respectively. It then deems that the difference between the two sample sets is an approximation of $R(P_i)$. Therefore,

$$R(P_i) \approx Diff\left(\{E_{i,1}, \ldots, E_{i,j}, \ldots, E_{i,|R|}\}, \{E'_{i,1}, \ldots, E'_{i,k}, \ldots, E'_{i,|R'|}\}\right) \tag{2}$$

Thus, the non-parametric hypothesis testing method uses the difference between the two sample sets to measure the differences in the sampled distributions of evaluation biases. The corresponding p-value of the non-parametric hypothesis testing method is used to replace the ranking function in Eq. (2). According to the corresponding ranking scores of the predicates, all the predicates are sorted to form a predicate list. The programmer may use this generated predicate list to locate faults in a program.

4. Research questions

To investigate the applicability of using non-parametric hypothesis testing model for fault localization, we design the following research questions:

*Q*1: Is normal distribution common in program spectra (and evaluation biases of predicates in particular)?The answer to this question relates to whether it is suitable to use parametric hypothesis testing methods such as SOBER [17] on the evaluation biases of predicates for fault localization. If it is not common for the evaluation biases of predicates be normally distributed, the assumption that the program spectra on predicates can be regarded as normal distributions cannot be well supported. It appears not rigorous enough to use parametric hypothesis testing methods on the evaluation biases of predicate for fault localization.

*Q*2: Is normal distribution common in the program spectra of the most fault-relevant predicates (and evaluation biases in particular)?Many fault-localization techniques (such as [8,10,12,14–17]) generate a predicate list, which sorts all the predicates in descending order of their fault relevance. For these techniques, the most fault-relevant predicates play an important role, since the effectiveness of each technique is mainly decided by the efficiency in locating such predicates in the given predicate lists. Therefore, we also investigate the normality of the most fault-relevant predicates. If the answer to question *Q*1 is *no*, and yet the answer to this question is *yes*, the use of parametric hypothesis testing methods in fault localization may be still acceptable.

*Q*3: Does the normality of evaluation biases of the most fault-relevant predicates correlate with the effectiveness of a non-parametric fault-localization technique?If the answers to both questions *Q*1 and *Q*2 are *no*, it appears unsuitable to uphold the assumption that the underlying program spectra form normal distributions. It also indicates that the use of a non-parametric fault-localization technique such as the one proposed in our previous work [8], is a viable choice. This leads us to further investigate whether the normality of evaluation biases of the most fault-relevant predicates correlates with the effectiveness of a non-parametric fault-localization technique.

5. Experiment

We present the experiment in this section. We first introduce the subject programs, the selected evaluation metric, and the selected normality test method. After introducing the research questions, we describe the setup of experiment. We report the result of the normality test on the evaluation bias of each predicate in each faulty version, and investigate the effect of the normality of predicates on fault-localization techniques. Based on the empirical data, we set up the hypotheses and use standard hypothesis testing method to answer the three research questions above. At the end of this section, we will discuss the threats to validity of our experiment.

5.1. Subject programs

The Siemens suite consists of 132 C programs with seeded faults. Each program is a variation of one of seven programs,

Table 1
Statistics of Siemens programs.

Siemens programs	No. of faulty versions	Executable LOC	Average no. of test cases
print_tokens	7	341–342	4130
print_tokens2	10	350–354	4115
replace	32	508–515	5542
schedule	9	291–294	2650
schedule2	10	261–263	2710
tcas	41	133–137	1608
tot_info	23	272–274	1052

namely "tcas", "tot_info", "replace", "print_tokens", "print_tokens2", "schedule", and "schedule2", varying in sizes from 133 to 515 executable lines.[1] Each faulty version is seeded with one fault. These programs and their faulty versions are downloaded from the Software-artifact Infrastructure Repository (SIR) [5] website. Table 1 shows the descriptive statistics of the suite, including the number of faulty versions (column "No. of Faulty Versions"), the number of executable lines of code (column "Executable LOC"), and the number of test cases (column "Average No. of Test Cases").

5.2. Evaluation metrics

In this section, we first introduce our performance metric, *P*-score, to measure effectiveness of fault-localization techniques. Then, we introduce a statistical normality test and discuss the use of its *p*-value as a metric of normality. Finally, we introduce the correlation relation metric.

5.2.1. Effectiveness metric: P-score

In previous studies [3,16,17,19], to gauge the quality of fault localization, *T*-score is used as the metric. It was originally proposed by Renieris and Reiss [19] and later adopted by Liu et al. [16,17] in reporting the performance of their fault-localization techniques.

The *T*-score metric helps measure the cost of locating a fault using a fault-localization technique. However, some limitations have been reported on the use of *T*-score in previous work [3,19]. (i) They claim that their evaluation setup "*assumes an ideal programmer who is able to distinguish defects from non-defects at each location, and can do so at the same cost for each location considered.*" [3]. (ii) Besides, *T*-score assumes that the developer can follow the control- and/or data-dependency relations among statements when searching for faults. However, there is no evidence that it resembles the manner of debugging in real life.

To better reflect the effectiveness of the non-parametric fault-localization technique, we propose a novel metric, which we call **P-score**, to evaluate them. We recall that many fault-localization techniques [10–12,16,17] (including the non-parametric fault-localization technique) generate a predicate list, which contains all the predicates sorted in descending order of their degree of fault relevance (in terms of how much each of them is deemed to be relevant to fault). Such degree of fault relevance is measured by the ranking formula of the technique. For postmortem analysis, we mark the predicate closest to any fault in the program, and use the position of the predicate in the predicate list as the indicator of the effectiveness of a fault-localization technique in generating the predicate list. We call such a predicate *the most fault-relevant predicate*. Suppose L is the predicate list and $\widetilde{P}$ is the most fault-relevant predicate. The measurement formula is given by Eq. (3). To ease our presentation, we simply call this metric the *P*-score

$$P\text{-}score = \frac{\text{1-based index of } \widetilde{P} \text{ in } L}{\text{number of predicates in } L} \times 100\% \qquad (3)$$

where a 1-based index means that the first element of L is indexed by 1 (rather than 0). The metric *P*-score reflects the effectiveness of a fault-localization technique. The lower the value, the more effective will be the fault-localization technique. For tie cases, which mean that there exist multiple most fault-relevant predicates on the same predicate list, we count $\widetilde{P}$ as the first one reached in L.

For example, the faulty version "v1" of program "schedule2" (from the Siemens programs) contains 43 predicates. The fault lies in line 135; the most fault-relevant predicate exists in line 136. Suppose a fault-localization technique ranks predicate $\widetilde{P} = P_{136}$ at the second top position in the generated predicate L. The *P*-score is calculated as $\frac{2}{43} \times 100\% \approx 4.65\%$.

5.2.2. Normality test: the Jarque–Bera test

To measure whether the evaluation biases of a predicate form a normal distribution, we adopt the standard normality test method, the Jarque–Bera test [18]. The Jarque–Bera test is used to test the null hypothesis that the given population is from a normal distribution. The *p*-value of the Jarque–Bera test is used to measure how much the evaluation biases of a predicate form a normal distribution. For example, a *p*-value less than 0.05 means that the null hypothesis can be rejected at the 0.05 significance level [18]. It also means that the probability of obtaining an observation agreeing with the null hypothesis (being normal distribution) is less than 0.05. In general, the smaller the *p*-value, the more confident we will be in rejecting the null hypothesis. In other words, the smaller the *p*-value, the farther will be the evaluation biases of the predicate from a normal distribution.

To help readers follow the idea of normality tests, we use three different populations to illustrate the outcomes of the Jarque–Bera test. We use histograms to represent the distributions of these three populations. The respective histograms are shown in Fig. 4. We observe that, among the three populations, the leftmost one (Fig. 4a) is closest to a normal distribution. The rightmost one (Fig. 4c) is farthest from a normal distribution. The central one (Fig. 4b) is in between the two scenarios. The result of the *p*-value for the population in Fig. 4a is 0.7028. It means that we have a 70.28% probability that the observed data in Fig. 4a is from a normally distributed population. The result of the *p*-value for the population in Fig. 4b is 0.2439. It means that, we have a 24.39% probability that the observed data in Fig. 4b is from a normally distributed population. The *p*-value of the population in Fig. 4c is 0.0940. According to the results of normality tests, we can determine that the population in Fig. 4a is closest to a normal distribution, followed by the population in Fig. 4b, while the population in Fig. 4c is farthest from a normal distribution. Thus, the normality test results match our expectation.

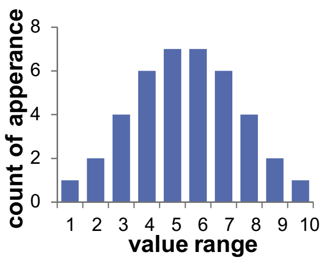
In the rest of this paper, we will use the results of the *p*-value in the Jarque–Best test as the degree of normality for predicates.

5.2.3. Correlation metrics: Pearson correlation test

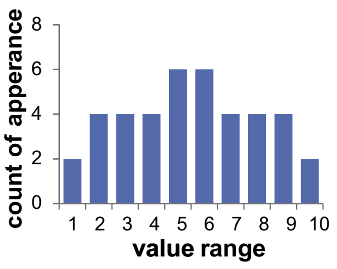
Pearson correlation test [18] is designed to evaluate the direction and strength of the linear relationship between two populations. The resulting Pearson correlation coefficient is in the range of $[-1, 1]$. The correlation coefficient is close to 1 in the case of an increasing linear relationship. It is close to -1 in the case of a decreasing linear relationship. If the two populations are independent of each other, the correlation coefficient is close to 0.

For example, we use three sets of data to illustrate the outcomes of the Pearson correlation test. The three sets are represented by the points in Fig. 5a–c, respectively. For each point, the *X*- and *Y*-coordinates stand for the values of the *X* and *Y* variables, respectively.

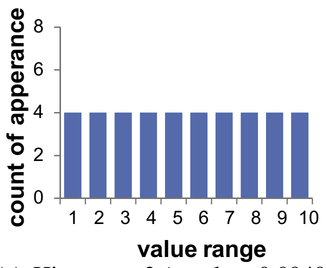
[1] We use the tool "David A. Wheeler's SLOCCount" to count the executable statements. It is available at http://www.dwheeler.com/sloccount/.



(a) Histogram 1 (p-value=0.7028)

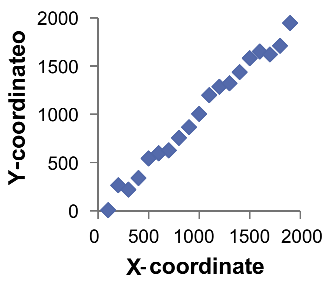


(b) Histogram 2 (p-value=0.2439)

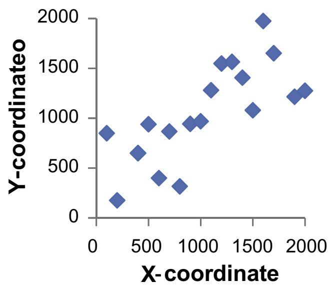


(c) Histogram 3 (p-value=0.0940)

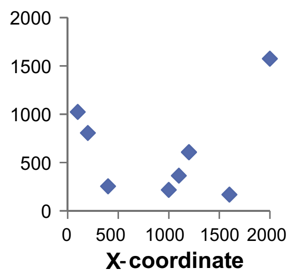
Fig. 4. Illustrations for normality tests.



(a) Data set 1 (0.995)



(b) Data set 2 (0.770)



(c) Data set 3 (0.173)

Fig. 5. Illustration of Pearson correlation test.

Let us first focus on the leftmost set of data (Fig. 5a). The *Y*-coordinate conforms to a linear increasing function of the *X*-coordinate. The Pearson correlation coefficient for this set of data is 0.995. For the rightmost set of data (Fig. 5c), the *X*- and *Y*-coordinates do not have strong dependence relationships between each other. The Pearson correlation coefficient for this set of data is 0.173. For the set of data in Fig. 5b, the situation is in between the two scenarios above. The corresponding correlation coefficient is 0.770. From these examples, we observe that the Pearson correlation test is useful in determining the linear relationship between two populations.

5.3. Setup of experiment

Among the 132 programs from the Siemens suite, two of them (version "v27" of program "replace" and version "v9" of program "schedule2") come with no failed test cases, which is also reported in previous studies [16,17]. These two versions are excluded in our experiment because our non-parametric fault-localization method needs the presence of both successful and failed test cases. To evaluate our method, we follow [16,17] to use the entire test suite as input to our method.

For each of the applicable faulty versions, we compare the program output with that of the corresponding original version, and mark the differences on the faulty versions as faulty statements. The most fault-relevant predicates are manually marked, by referring to their relative distances from the faulty statements. For 111 out of 130 faulty versions, it is straightforward to identify the most fault-relevant predicates. In each of these faulty versions, the distance between the most fault-relevant predicate and the faulty statement is less than three program lines. For each of the remaining 19 faulty versions, the faulty statement lies in a function module without any predicate. Since it is hard to identify the most fault-relevant predicate in such situations, we exclude them in our experiment. These faulty versions include versions "v4", "v5", and "v6" of program "print_tokens", version "v12" of program "replace", versions "v13", "v14", "v16", "v17", "v18", "v19", "v33", "v36", "v38", "v7", and "v8" of program "tcas", and versions "v10", "v12", "v21", and "v6" of program "tot_info".

Our experiment is carried out on a Dell PowerEdge 1950 server with two 4-core Xeon 5355 (2.66 Hz) processors. The operating system is Solaris Unix with the kernel version Generic 120012-14. Our experimental platform is constructed using the tools of flex++ 2.5.31, bison++ 1.21.9-1, CC 5.8, bash 3.00.16(1)-release (i386-pc-solaris2.10), and sloccount 2.26. The source codes for the Mann–Whitney test and the Jarque–Bera normality test are from the ALGlib website.[2]

In total, we collect the evaluation biases of 5778 predicates from all 111 faulty versions. In the next three subsections, we are going to discuss the three research questions that have been stated in the last section.

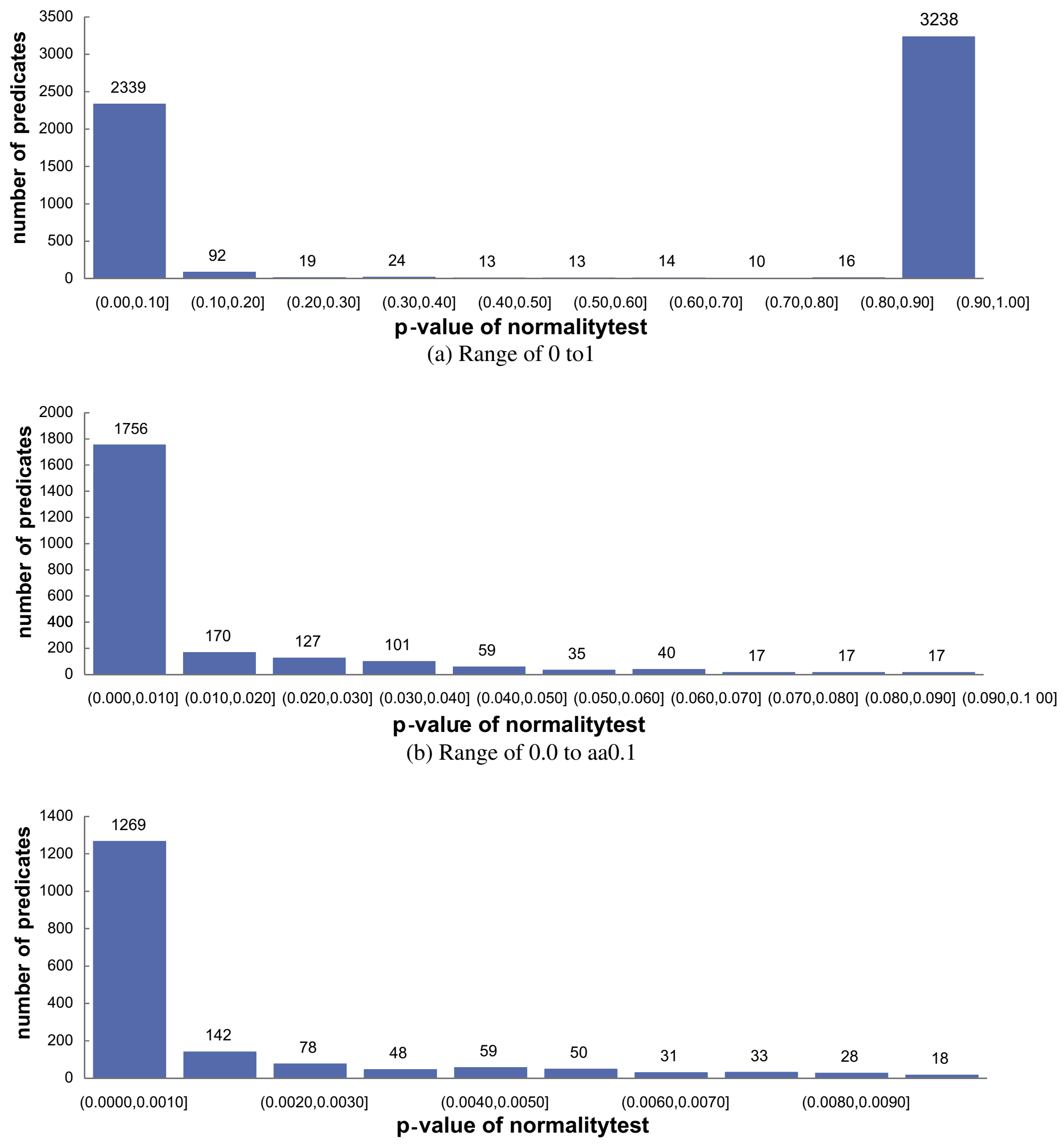
5.4. Answering research question Q1

In this section, we first conduct normality tests on the evaluation biases of the 5778 predicates from the 111 faulty versions of Siemens programs, and report the distributions of the evaluation biases of these predicates. After that, we use hypothesis testing method to answer Q1.

5.4.1. Direct observations

The distributions (represented by histograms) of the normality test results are shown in Fig. 6. For each predicate, we separately consider its evaluation biases in passed runs and those in failed runs. For each predicate, we specify its normality test result as the minimum of the normality test results of evaluation biases in all passed runs and all failed runs. Fig. 6a shows a plot of the numbers of predicates against given *p*-values of the normality tests of their evaluation biases, in the range of [0,1] (10 segments). The

[2] Available at http://www.alglib.net/.



(a) Range of 0 to1

(b) Range of 0.0 to aa0.1

(c) Range of 0.00 to 0.01

Fig. 6. Results of normality tests for predicates.

leftmost data column stands for the predicates having *p*-values less than or equal to 0.1. The number of such predicates is 2399. It means that, if the null hypothesis (that evaluation biases of predicates form normal distributions) is true, for 2399 predicates, the probability of the appearance of their observed evaluation biases is less than 10%. In other words, for these 2399 predicates, the null hypothesis can be rejected at the 10% significance level. The rightmost column stands for the predicates having *p*-values greater than 0.9. The number of such predicates is 3238. It means that, if the null hypothesis is true, for these 3238 predicates, the probability of observing the sample evaluation biases is higher than 90%. The eight columns at the center show the predicates having *p*-values in the ranges of (0.1, 0.2], (0.2, 0.3], (0.3, 0.4], (0.4, 0.5], (0.5, 0.6], (0.6, 0.7], (0.7, 0.8], and (0.8, 0.9], respectively. These ranges are decided by uniformly dividing the range of [0, 1]. The second and plots show the same statistics in the ranges of [0.0, 0.1] and [0.00, 0.01], respectively.

From Fig. 6a, we observe that not all of the predicates form statistically meaningful normal distributions. If we choose 0.1 as the significance level for the *p*-values of normality tests, the null hypothesis can be rejected for 2339 predicates (or more than 40%). We deem their evaluation biases to be far from having normal distributions. If we deem 0.9 as the significance level for the *p*-values of normality tests, 3238 predicates (or less than 60%) are recognized to have normally distributed evaluation biases. We deem that their evaluation biases have normal distributions.

Fig. 6b is a zoom-in representation of the range of [0.0, 0.1] (the range of the leftmost data column of Fig. 6a). We notice that, for 1756 out of 2339 predicates (or more than 75%) having *p*-values in the range of [0.0, 0.1], their corresponding *p*-values concentrate in the range of [0.00, 0.01]. Let us focus on Fig. 6c. We further notice that, for 1269 out of 1756 predicates (or close to 75%) having *p*-values in the range of [0.00, 0.01], their *p*-values concentrate in the range of [0.000, 0.001].

Our observations are that: (i) For no more than 60% of the predicates, their evaluation biases have normal distributions (the null hypothesis cannot be rejected at less than a significance level of 0.9). (ii) For a majority of the remaining 40% of the predicates, their evaluation biases are far from having normal distributions (the null hypothesis can be rejected at the 0.1 significance level). (iii) There are few predicates whose degrees of normality are within the range of [0.1, 0.9].

5.4.2. Hypothesis testing: answering Q1

Readers may recall that we make use of the standard normality test method, the Jarque–Bera test, to measure the *degree of normality* for the evaluation biases of predicates. The higher the degree of normality for a predicate (in terms of the *p*-value of the Jarque–Bera test), the closer will be the observed evaluation biases to a normal distribution. Since we calculate a degree of normality for each predicate, we design the following null hypothesis to answer research question Q1:

*H*1: "*The mean degree of normality for the tested predicates is greater than a given threshold* θ_1."

Such a null hypothesis captures the *mean* degree of normality for the predicates, and is therefore used to determine whether normal distributions are *common* for predicates. In addition, we introduce a parameter θ_1 to control the power of the null hypothesis. The higher the value of θ_1, the more confidence will we have on the null hypothesis. Hence, for the same set of predicates, the higher the value chosen for θ_1, the easier will the null hypothesis be rejected.

To answer research question *Q*1, we conduct the one-tail Student's *t*-test [29] to validate *H*1. The *p*-value of the Student's *t*-test is the probability that an observed predicate from a population with a mean degree of normality is greater than θ_1. Suppose, for instance, that we have only three predicates with degrees of normality 0.640, 0.750, and 0.860, respectively. The *p*-value of the Student's *t*-test on null hypothesis *H*1 with $\theta_1 = 0.600$ will be 0.929. On the other hand, the *p*-value of the Student's *t*-test on null hypothesis *H*1 with $\theta_1 = 0.750$ will be 0.500. Similarly, the *p*-value of the Student's *t*-test on null hypothesis *H*1 with $\theta_1 = 0.900$ will be 0.071. From the first *p*-value, the probability that the predicates are from a population with a mean degree of normality greater than 0.600 is 92.9%. The other two *p*-values can be interpreted similarly.

We vary the value of θ_1 within the range of [0, 1]. The corresponding *p*-values of one-tail Student's *t*-tests are shown in Table 2. The upper row shows the threshold values for θ_1, while the lower row shows the results of the one-tail Student's *t*-tests in terms of *p*-values.

We observe that we have great confidence (with a probability close to 100.0%) that these predicates are from a population with a mean degree of normality greater than 0.500. At the same time, the probability that these predicates are from a population with a mean degree of normality greater than 0.600 is less than 0.01%. Hence, from the meaning of the null hypothesis and the symmetry of the one-tail test, we conclude that it is very likely (with a probability close to 100.0%) that these predicates are from a population with a mean degree of normality in the range of [0.500, 0.600].

We note that in order to be statistically significant (in rejecting a null hypothesis), it generally requires at least a significance level of 0.1 [29]. Since we want to study the normality of program spectra in a conservative manner, we set the threshold of the degree of normality to a reasonable value (such as 0.700, 0.800, or higher) in the above null hypothesis. With $\theta_1 > 0.600$, the null hypothesis *H*1 can always be rejected at the 0.0001 significance level (the resultant *p*-value is less than 0.0001). Obviously, 0.0001 is a reasonably small value for significance levels; and we hence conclude that normal distributions are *not* common for the evaluation biases of predicates. The answer to Q1 is *no*.

We know that the smaller the value of the threshold, the more difficult it will be to reject a null hypothesis. Suppose, for example, that we choose a value of 0.600 as the threshold. The hypothesis becomes: "*The mean degree of normality for the tested predicates is greater than 0.600*". Still, such a null hypothesis can continue to be rejected at the 0.0001 significance level. Our result, as presented in Table 2, indicates that many predicates cannot produce dynamic behaviors that form normal distributions even if one wishes to lower the judgment standard to wishfully assume that it could be the case.

Table 2
Student's *t*-test on different thresholds for *H*1.

Threshold value (θ_1)	0.000–0.500	0.576	0.584	0.587	0.591	0.600–1.000
p-Value	1.000	0.500	0.100	0.050	0.010	$\leqslant 0.0001$

5.5. Answering research question Q2

In this section, we first conduct normality tests on the evaluation biases of the most fault-relevant predicates from the 111 faulty versions of Siemens suite, and report their normality distributions. After that, we use the hypothesis testing method to study Q2. At the end of this subsection, we compare the statistical natures of the most fault-relevant predicates with the other predicates.

5.5.1. Direct observations

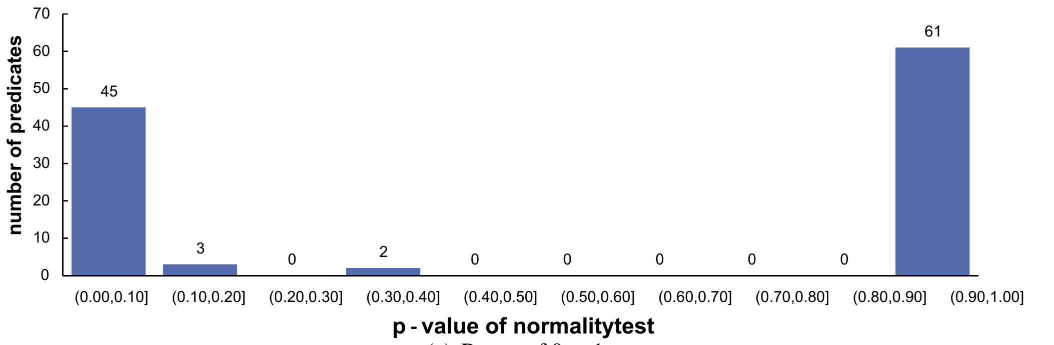
We know from the previous subsection that not all the predicates can be regarded as having normally distributed evaluation biases. One may wonder whether parametric techniques may reasonably be applied assuming that the dynamic behaviors of the most fault-relevant predicates are close to normal distributions. In this subsection, therefore, we are going to study whether the evaluation biases of the most fault-relevant predicates may exhibit normal distributions. Like the analysis in the previous subsection, the same normality test method is applied to the most fault-relevant predicates (111 in total). For each of these predicates, we separately consider its evaluation biases in passed runs and those in failed runs. Furthermore, for each predicate, we specify its normality test result as the minimum among its normality test results of evaluation biases in all passed runs and those in all failed runs.

The results are shown in Fig. 7. It shows the distributions of *p*-values for normality tests on the 111 most fault-relevant predicates in the faulty versions. It can be interpreted similarly to Fig. 6.

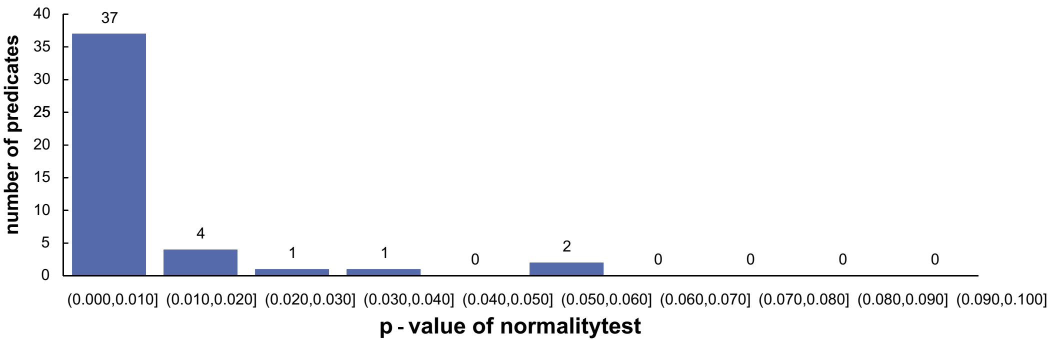
Let us focus on Fig. 7a first. If we deem 0.9 as the significance level for the *p*-values of normality tests, 61 out of 111 most fault-relevant predicates (less than 55%) are recognized as exhibiting normally distributed evaluation biases. It means that, if the null hypothesis is true, for 61 predicates, there is more than 90% probability for their observed evaluation biases to appear. At the other extreme, if we choose 0.1 as the significance level (which is the de facto practice in hypothesis testing), there are still 45 predicates (more than 40%) that the null hypothesis can be rejected.

We observe from Fig. 7b that, for 37 out of 45 predicates (more than 80%) having corresponding *p*-values in the range of [0.0, 0.1], these *p*-values concentrate in the range of [0.00, 0.01]. When we zoom in further, as shown in Fig. 7c, for 31 out of 37 predicates (close to 85%) whose *p*-values are in the range of [0.00, 0.01], their *p*-values concentrate in the range of [0.000, 0.001].

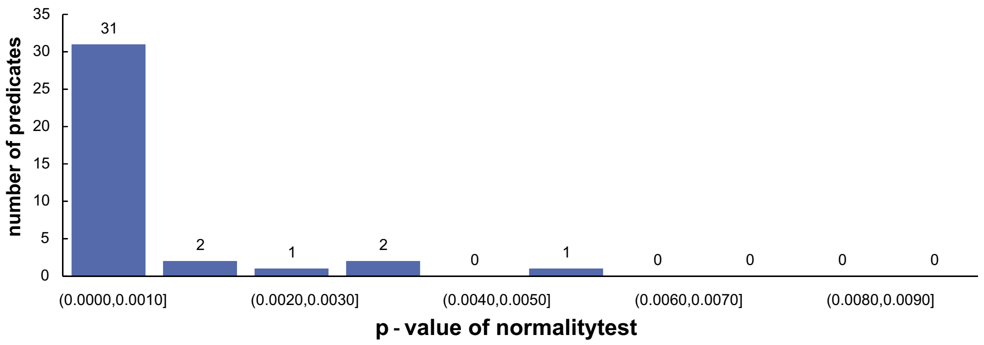
Our observations are summarized as follows: (i) For about 55% of the predicates, their evaluation biases have normal distributions (the null hypothesis cannot be rejected at a significance level of 0.9). (ii) For about 40% of the predicates, their evaluation biases are far from exhibiting normal distributions (the null hypothesis can be rejected at the 0.1 significance level). (iii) There are about 5% of the predicates whose normality test results are within the range of [0.1, 0.9].



(a) Range of 0 to 1



(b) Range of 0.0 to 0.1



(c) Range of 0.00 to 0.01

Fig. 7. Results of normality tests for the most fault-relevant predicates.

5.5.2. Comparison with other predicates

We further compare the statistics of the most fault-relevant predicates with the statistics of all the predicates collected in the experiment. The results are shown in Table 3. Take the leftmost data column as an example. It means that only 54.95% (61 out of 111) of the most fault-relevant predicates have *p*-values greater than 0.900, and only 56.04% (3238 out of 5778) of all the predicates studied have *p*-values of normality tests greater than 0.900. In other words, only about 55% of the most fault-relevant predicates and 56% of all predicates exhibit a normal distribution at the 90% significance level. Since parametric statistical techniques assume a normal distribution of evaluation biases of predicates, the application of such techniques to every predicate is questionable because nearly half of the predicates do not satisfy the underlying assumption. If the precondition cannot be satisfied, the conclusion from the precondition cannot, of course, be established. This finding indicates the reason why existing parametric statistical fault-localization techniques do not work as robustly as non-parametric techniques.

Table 3
Comparison of statistics of predicates with statistics of the most fault-relevant predicates.

Range of *p*-values for normality test	>0.900	>0.500	>0.100	>0.050	>0.010
% of the most fault-relevant predicates (actual number)	54.95 (61)	54.95 (61)	59.46 (66)	61.26 (68)	66.67 (74)
% of all predicates (actual number)	56.04 (3238)	58.69 (3391)	59.52 (3439)	61.70 (3565)	78.04 (4509)

5.5.3. Hypothesis testing: answering Q2

Similarly to Q1, we design the following null hypothesis to answer research question Q2:

> H2: “*The mean degree of normality for the most fault-relevant predicates under test is greater than a given threshold* θ_2.”

Such a null hypothesis captures the *mean* degree of normality for the most fault-relevant predicates, and is therefore used to determine whether normal distributions are *common* for the most fault-relevant predicates. A parameter θ_2 is introduced to control

Table 4
Student's *t*-tests on different thresholds for *H*2.

Threshold value (θ_2)	0.000–0.400	0.561	0.621	0.638	0.671	0.700–1.000
p-Value	1.000	0.500	0.100	0.050	0.010	$\leqslant$ 0.0001

the power of the null hypothesis. The higher the value of θ_2, the more confidence will we have on the null hypothesis. Hence, for the same set of predicates, the higher the value chosen for θ_2, the easier will the null hypothesis be rejected.

To answer research question *Q*2, we also conduct the one-tail Student's *t*-test [29] to validate *H*2. The *p*-value of the Student's *t*-test is the probability that an observed predicate from a population with a mean degree of normality is greater than θ_2. We vary the value of θ_2 within the range of [0,1]. The corresponding *p*-values of one-tail Student's *t*-tests are shown in Table 4. The upper row shows the threshold values for θ_2, while the lower row shows the results of the one-tail Student's *t*-tests in terms of *p*-values.

We have great confidence (with a probability close to 100.0%) that these predicates are from a population with mean degree of normality greater than 0.400. At the same time, the probability that these predicates are from a population with a mean degree of normality greater than 0.700 is less than 0.01%. Therefore, from the meaning of the null hypothesis and the symmetry of the one-tail test, we draw the conclusion that it is very possible (with a probability close to 100.0%) that these predicates are from a population with the mean degree of normality in the range of [0.400, 0.700].

Similarly, since we want to study the normality of program spectra in a conservative manner, we set the threshold of the degree of normality to a reasonable value (such as 0.700, 0.800, or higher) in the above null hypothesis. With $\theta_2 > 0.700$, the null hypothesis *H*2 can always be rejected at the 0.0001 significance level (the resultant *p*-value is less than 0.0001). Since 0.0001 is a reasonably small value as a significance level, we conclude that normal distributions are *not* common for the evaluation biases of the most fault-relevant predicates. The answer to *Q*2 is *no*. And our results as presented in Table 4 indicates that many fault-relevant predicates cannot produce dynamic behaviors that form normal distributions even if one wishes to lower the standard of judgment to wishfully assume that it could be the case.

5.6. Answering research question Q3

In this section, we first report the findings and then analyze the results using hypothesis testing to answer *Q*3.

5.6.1. Direct observations

From previous subsections, we know that the assumption of evaluation biases of predicates forming normal distributions is not well supported by the experiment on the Siemens suite. Since the non-parametric fault-localization technique is supposedly not based on such an assumption, we predict that the effectiveness of non-parametric fault-localization technique does not correlate with the normality of predicates. Fig. 8 gives the results of the corresponding correlation tests. To investigate whether it is the case, we analyze the *P*-score of the predicate list produced by our fault-localization technique against the degree of normality.

Fig. 8 depicts the correlations between the *p*-values of the most fault-relevant predicates and the results of our non-parametric fault-localization technique. In this figure, there are 111 points, which stand for the 111 faulty versions. The *X*-coordinates show the *p*-values of normality tests for the most fault-relevant predicates. The *Y*-coordinates show the *P*-scores for the same faulty version. This figure is divided into two parts. The left rectangle

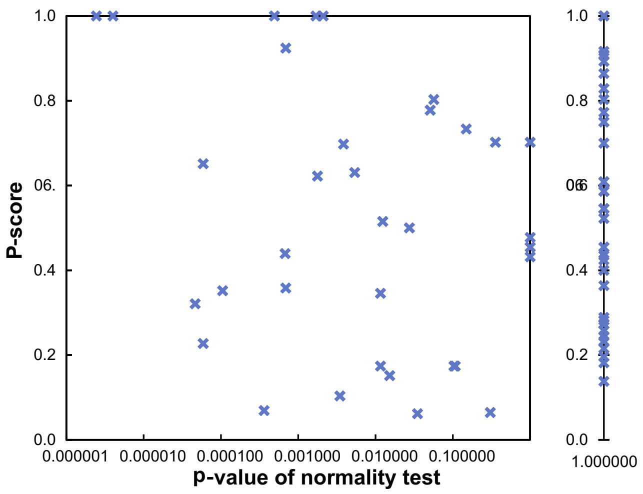


Fig. 8. Effect of normality on fault-localization techniques.

represent the 55 most fault-relevant predicates with *p*-values less than 1.0. The right axis represents the 56 most fault-relevant predicates with *p*-values equal to 1.0.

We observe that, as the *p*-value changes from 0.0 to 1.0, the *P*-scores of the fault-localization technique on the faulty versions do not show an obvious increasing or decreasing trend. On the contrary, the *P*-scores appear scattered across the whole rectangle. Apparently, as far as our non-parametric fault-localization technique is concerned, the normality of evaluation biases of the most fault-relevant predicate does not strongly correlate with the effectiveness of the technique in locating faults.

5.6.2. Hypothesis testing: answering Q3

We design the following hypothesis to answer *Q*3:

*H*3: "*The correlation of normality and effectiveness is greater than a given threshold* θ_3."

To scientifically investigate the effect of normality on the fault-localization techniques, we conduct the Pearson correlation test on them. The Pearson correlation coefficient on the *P*-score of the non-parametric fault-localization technique on the 111 faulty versions is 0.1201. If we only count the 55 faulty versions whose *p*-values are less than 1.0, the Pearson correlation coefficient on the *P*-score of the non-parametric fault-localization technique is 0.0037.

We may choose some reasonable threshold values (such as 0.700, 0.750, or higher) to determine whether there exist strong correlations between the degree of normality of the most fault-relevant predicate from a faulty version and the effectiveness of fault-localization techniques on the same faulty version. However, the hypothesis *H*3 with $\theta_3 \geqslant 0.700$ can always be rejected easily. It means that, the answer to *Q*3 is *no*, and the normality for evaluation biases of predicates only *weakly* correlates with the effectiveness of the non-parametric fault-localization technique. This can also indicate that the non-parametric hypothesis testing model for fault localization has high robustness in terms of the normality for the evaluation biases of predicates.

5.7. Threats to validity

In this section, we discuss the threats to internal, construct, and external validity of our experiment.

5.7.1. Internal validity

Internal validity is related to the cause and effect of the experimental findings.

We design null hypotheses and use hypothesis testing to answer research questions. To control the power of the designed null hypothesis, however, some parameters are involved in the research questions *Q*1, *Q*2, and *Q*3. The arbitrary choice of these parameters does not have scientific support. To address this threat, we adopt some value widely used previously (such as 0.700 for correlation tests), or change the values within a reasonable range to conduct hypothesis testing several times with different thresholds.

Many statistical algorithms are involved in our experiment, including the Mann–Whitney test, the Jarque–Bera test, and the Student's *t*-test. Different implementation details (such as accuracy in floating-point operations) may affect the experimental results. To address this threat, we choose to use the same programming library (ALGlib) to implement these algorithms. Such a consideration can reduce implementation faults and hence the computing errors.

5.7.2. Construct validity

Construct validity lies in the evaluation method we choose.

Although *T*-score is widely used in previous work (including [16,17]), some limitations have also been reported in its use (see [3], for example). Has any other measures been used to evaluate predicate-based techniques successfully? We are not aware of such alternatives in the public literature. Therefore, we use a novel metric, *P*-score. The *P*-score metric evaluates the speed of locating the most fault-relevant predicate using the generated predicate list. The consideration is that all these techniques estimate the fault relevance of predicates and generate a list of predicates according to their suspiciousness.

We use the Mann–Whitney test, the Jarque–Bera test, the Student's *t*-test, and the Pearson correlation test in the experiment. Using other kinds of hypothesis testing methods, normality test methods, or correlation test methods may produce different results. To address this issue, all the methods we choose are representative among their respective families.

Threats may also exist in the manual work involved in the experiments, since we manually mark the most fault-relevant predicates. This step is neither entirely objective nor automated. However, for faulty versions whose most fault-relevant predicates are hard to identify, we have excluded them from the experiment to avoid subjective biases. How to decide the most fault-relevant predicates in programs can be further explored in future work.

5.7.3. External validity

External validity is related to whether the results of the experiment can be generalized.

Our experiment is conducted on the Siemens programs. Though the Siemens programs are representative programs and have been used in many previous studies [10,12,17,16], the use of other programs (such as JAVA programs instead of C programs) may product different results. In addition, the use of different normality tests or hypothesis testing methods may also affect the experimental results. Evaluations on the use of other alternatives should be included in future work.

6. Related work

Program slicing [22] is a code-based technique. It is widely used in debugging [20]. Gupta et al. [6] propose a forward dynamic slicing approach to narrow down slices. They further integrate the forward approach with standard dynamic slicing approaches [26].

Collofello and Cousins [4] pioneer the use of test cases for fault localization. A promising approach is to use the behavioral statistics collected from test case executions. *Delta debugging* helps to simplify or iron out fragments of failed test cases [25], producing cause–effect chains [24] and linking them to suspicious statements [3].

Harrold et al. [7] list nine classes of program spectra, such as path count, data-dependency count, and execution trace. Among them, the execution trace spectrum is most widely used in debugging. Jones et al. [10,12], in their work *Tarantula*, rank each statement according to suspiciousness, which is a function of the percentages of failed and successful test cases that execute the statement. Renieris and Reiss [19], in their work *Nearest Neighbor*, find that the execution trace difference between a failed run and its nearest passed neighboring run is more effective for debugging. Baudry et al. [2] observe that some statements (known as a dynamic basic block) are always executed by the same set of test cases. They use a bacteriologic approach to generate test cases so as to maximize the number of dynamic basic blocks, and use the algorithm in [10,12] to rank them. They further extend their work in [11] to make it possible for multiple developers to debug at the same time.

The most relevant related projects are CBI [14,15] and SOBER [16,17]. Rather than locating faulty statements, these techniques make use of predicates to indicate the fault locations. Since these techniques have been explained in Section 1, we do not repeat them here. Arumuga Nainar et al. [1] further extend CBI to address compound Boolean expressions. Zhang et al. [28] propose a fine-grained version of such techniques and use an empirical study to investigate the effectiveness. Wong et al. [23] propose a code coverage-based fault-localization technique, which uses a utility function to calibrate the contribution of each passed run when calculating the fault relevance of executed statements.

In our previous work [8], we observe that the assumption of evaluation biases of predicates forming a normal distribution is not well supported by empirical data, and accordingly propose a non-parametric hypothesis testing model for fault localization. However, the investigation about normality of predicate and its effect on fault localization are not included. Zheng et al. [27] incorporate the notion of error propagation along with program dependences to statistical debugging. Wang et al. [21] formulate the notion of context patterns to improve statistical debugging. Jiang et al. [9] empirically study the relationship between statistical fault localization and test case prioritization and find that random ordering is superior to many test case prioritization techniques.

7. Conclusion

Fault localization is a time-consuming and yet crucial activity in software debugging. Many previous studies contrast the feature spectra between passed runs and failed runs to locate the predicates related to faults (or to locate the faulty statements directly). For instance, CBI [14,15] and SOBER [16,17] contrast the feature spectra of predicates in a program. They collect statistics about the behaviors of program predicates, such as evaluations of branch statements. They further assume that, for predicates near the fault position, the successes or failures of their evaluations highly correlate with the successes or failures of program executions. Hence, identifying effective program predicates and formulating correct and robust statistic comparisons are important for such techniques. However, they overlook the investigation of the statistical distributions of the spectra, on which parametric hypothesis testing methods fully or robustly rely. In our previous work, we have argued and empirically verified that assuming a specific distribution of the feature spectra of dynamic program statistics is problematic, and proposed a non-parametric approach that applies general hypothesis testing techniques to statistical fault localization.

In this paper, we rigorously investigate the statistical nature of normality test results for predicates. We show empirically that the evaluation biases of predicates, particularly the most fault-relevant predicates, are not necessarily normally distributed. In fact, almost all examined distributions of evaluation biases are either normal or far from normal, but not in between. Besides, the most fault-relevant predicates are less likely to exhibit normal distributions in their evaluation biases than other predicates. Unfortunately, the position of the most fault-relevant predicate(s) in a faulty program cannot be known before successful localization of the corresponding fault. Thus, to apply a hypothesis testing technique across the board, there is no scientific ground to assume all predicates to exhibit normal distributions among test executions.

We further investigate the effect of normality of predicates on fault-localization techniques, and use it as a measure to test the robustness of non-parametric fault-localization techniques. The empirical results show that the non-parametric model for fault localization has high robustness in terms of the normality for the evaluation biases of predicates. Our work also highlights a threat to construct validity in fault-localization techniques that employ parametric hypothesis testing methods.

Future work may include concurrent debugging of multi-fault programs and evolving software. It is also necessary to conduct more studies on large size programs. There are also an increasing number of data mining approaches to addressing software engineering issues. Our work also highlights the need to study potential normality assumptions in such mining approaches.

References

[1] P. Arumuga Nainar, T. Chen, J. Rosin, B. Liblit, Statistical debugging using compound Boolean predicates, in: Proceedings of the 2007 ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2007), ACM Press, New York, NY, 2007, pp. 5–15.

[2] B. Baudry, F. Fleurey, Y. Le Traon, Improving test suites for efficient fault localization, in: Proceedings of the 28th International Conference on Software Engineering (ICSE 2006), ACM Press, New York, NY, 2006, pp. 82–91.

[3] H. Cleve, A. Zeller, Locating causes of program failures, in: Proceedings of the 27th International Conference on Software Engineering (ICSE 2005), ACM Press, New York, NY, 2005, pp. 342–351.

[4] J.S. Collofello, L. Cousins, Towards automatic software fault location through decision-to-decision path analysis, in: Proceedings of the 1987 National Computer Conference, Chicago, IL, 1987, pp. 539–544.

[5] H. Do, S.G. Elbaum, G. Rothermel, Supporting controlled experimentation with testing techniques: an infrastructure and its potential impact, Empirical Software Engineering 10 (4) (2005) 405–435.

[6] N. Gupta, H. He, X. Zhang, R. Gupta, Locating faulty code using failure-inducing chops, in: Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering (ASE 2005), ACM Press, New York, NY, 2005, pp. 263–272.

[7] M.J. Harrold, G. Rothermel, K. Sayre, R. Wu, L. Yi, An empirical investigation of the relationship between spectra differences and regression faults, Software Testing, Verification and Reliability 10 (3) (2000) 171–194.

[8] P. Hu, Z. Zhang, W.K. Chan, T.H. Tse, Fault localization with non-parametric program behavior model, in: Proceedings of the 8th International Conference on Quality Software (QSIC 2008), IEEE Computer Society Press, Los Alamitos, CA, 2008, pp. 385–395.

[9] B. Jiang, Z. Zhang, T.H. Tse, T.Y. Chen, How well do test case prioritization techniques support statistical fault localization, in: Proceedings of the 33rd Annual International Computer Software and Applications Conference (COMPSAC 2009), IEEE Computer Society Press, Los Alamitos, CA, 2009.

[10] J.A. Jones, M.J. Harrold, Empirical evaluation of the Tarantula automatic fault-localization technique, in: Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering (ASE 2005), ACM Press, New York, NY, 2005, pp. 273–282.

[11] J.A. Jones, M.J. Harrold, J.F. Bowring, Debugging in parallel, in: Proceedings of the 2007 ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2007), ACM Press, New York, NY, 2007, pp. 16–26.

[12] J.A. Jones, M.J. Harrold, J. Stasko, Visualization of test information to assist fault localization, in: Proceedings of the 24th International Conference on Software Engineering (ICSE 2002), ACM Press, New York, NY, 2002, pp. 467–477.

[13] B. Korel, PELAS: program error-locating assistant system, IEEE Transactions on Software Engineering 14 (9) (1988) 1253–1260.

[14] B. Liblit, A. Aiken, A.X. Zheng, M.I. Jordan, Bug isolation via remote program sampling, in: Proceedings of the 2003 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2003), ACM SIGPLAN Notices 38 (5) (2003) 141–154.

[15] B. Liblit, M. Naik, A.X. Zheng, A. Aiken, M.I. Jordan, Scalable statistical bug isolation, in: Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2005), ACM SIGPLAN Notices 40 (6) (2005) 15–26.

[16] C. Liu, J. Han, Failure proximity: a fault localization-based approach, in: Proceedings of the 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering (SIGSOFT 2006/FSE-14), ACM Press, New York, NY, 2006, pp. 46–56.

[17] C. Liu, X. Yan, L. Fei, J. Han, S.P. Midkiff, SOBER: statistical model-based bug localization, in: Proceedings of the Joint 10th European Software Engineering Conference and 13th ACM SIGSOFT International Symposium on Foundation of Software Engineering (ESEC 2005/FSE-13), ACM SIGSOFT Software Engineering Notes 30 (5) (2005) 286–295.

[18] R. Lowry, Concepts and Applications of Inferential Statistics, Vassar College, Poughkeepsie, NY, 2006. Available at <http://faculty.vassar.edu/lowry/webtext.html>.

[19] M. Renieris, S.P. Reiss, Fault localization with nearest neighbor queries, in: Proceedings of the 18th IEEE International Conference on Automated Software Engineering (ASE 2003), IEEE Computer Society Press, Los Alamitos, CA, 2003, pp. 30–39.

[20] F. Tip, A survey of program slicing techniques, Journal of Programming Languages 3 (3) (1995) 121–189.

[21] X. Wang, S.C. Cheung, W.K. Chan, Z. Zhang, Taming coincidental correctness: refine code coverage with context pattern to improve fault localization, in: Proceedings of the 31st International Conference on Software Engineering (ICSE 2009), IEEE Computer Society Press, Los Alamitos, CA, 2009, pp. 45–55.

[22] M. Weiser, Program slicing, IEEE Transactions on Software Engineering SE-10 (4) (1984) 352–357.

[23] W.E. Wong, Y. Qi, L. Zhao, K.-Y. Cai, Effective fault localization using code coverage, Proceedings of the 31st Annual International Computer Software and Applications Conference (COMPSAC 2007), vol. 1, IEEE Computer Society Press, Los Alamitos, CA, 2007, pp. 449–456.

[24] A. Zeller, Isolating cause–effect chains from computer programs, in: Proceedings of the 10th ACM SIGSOFT International Symposium on Foundations of Software Engineering (SIGSOFT 2002/FSE-10), ACM SIGSOFT Software Engineering Notes 27 (6) (2002) 1–10.

[25] A. Zeller, R. Hildebrandt, Simplifying and isolating failure-inducing input, IEEE Transactions on Software Engineering 28 (2) (2002) 183–200.

[26] X. Zhang, N. Gupta, R. Gupta, Locating faults through automated predicate switching, in: Proceedings of the 28th International Conference on Software Engineering (ICSE 2006), ACM Press, New York, NY, 2006, pp. 272–281.

[27] Z. Zhang, W.K. Chan, T.H. Tse, B. Jiang, X. Wang, Capturing propagation of infected program states, in: Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT International Symposium on Foundation of Software Engineering (ESEC 2009/FSE-17), ACM Press, New York, NY, 2009.

[28] Z. Zhang, B. Jiang, W.K. Chan, T.H. Tse, Debugging through evaluation sequences: a controlled experimental study, in: Proceedings of the 32nd Annual International Computer Software and Applications Conference (COMPSAC 2008), IEEE Computer Society Press, Los Alamitos, CA, 2008, pp. 128–135.

[29] D.G. Zill, M.R. Cullen, Advanced Engineering Mathematics, Jones and Bartlett Publishers, Sudbury, MA, 2006.