Enhance Fault Localization using a 3D Surface Representation*

Qiong Shi
School of Electronics and Computer Science and Technology
North University of China
Taiyuan, China
Joan.ShiQ@gmail.com

Zhenyu Zhang
State Key Laboratory of Computer Science
Institute of Software Chinese Academy of Sciences
Beijing, China
zhangzy@ios.ac.cn

Zhifang Liu
School of Computer Science and Technology
Beihang University
Beijing, China
iuma29@gmail.com

Xiaopeng Gao
School of Computer Science and Technology
Beihang University
Beijing, China
gxp@buaa.edu.cn

Abstract — **Debugging is a difficult and time-consuming task in software engineering. To locate faults in programs, a statistical fault localization technique makes use of program execution statistics and employs a suspiciousness function to assess the relation between program elements and faults. In this paper, we develop a novel localization technique by using a 3D surface to visualize previous suspiciousness functions and using fault patterns to enhance such a 3D surface. By clustering realistic faults, we determine various fault patterns and use 3D points to represent them. We employ spline method to construct a 3D surface from those 3D points and build our suspiciousness function. Empirical evaluation on a common data set, Siemens suite, shows that the result of our technique is more effective than four existing representative such techniques.**

Keywords — data visualization, fault localization, data mining

I. INTRODUCTION

Although software is widely used in our daily life, it is far from bug-free. Software bugs in the famous Microsoft product, Xbox 360, have cost Microsoft more than USD 1 billion in the last years.

Software debugging is always an important but costly task in software engineering. Conventionally, debugging consists of three phases, that is, fault localization, fault repair, and re-test of corrected program. Many studies show that among the three phases, fault localization is the most difficult and time-consuming [10][12].

To locate faults in programs, statistical methods are used in previous studies [1][3][12][15]. They, so called statistical fault localization techniques, analyze program execution information, to assess the suspiciousness of a program element to be a fault. For example, if a statement is always exercised (covered) in failed executions, but never exercised in passed executions, it is very likely to be related to fault (or even, it is the fault). In a statistical fault localization framework, a suspiciousness function is often used to evaluate the relations among program elements and faults.

We find that in the fault localization problem settings, the values of the suspiciousness function is only related to two variables, the number of failed executions that exercise a program element and the number of passed executions that exercise a program element. We therefore use a 3D surface representation to visualize the suspiciousness function and employ it to investigate the properties of previous fault localization techniques. We find that realistic faults often have some fault patterns and such fault patterns can be used to enhance suspiciousness functions. For example, many faults existing in assignment statements are close to the main procedure and function entrance. Such statements are exercised in almost all the passed executions and all the failed executions. However, in the 3D surface, the corresponding region was given lower suspiciousness values in previous techniques. Calibrating the shape of the surface in this region may make a technique particularly effective for a majority of such kind of faults.

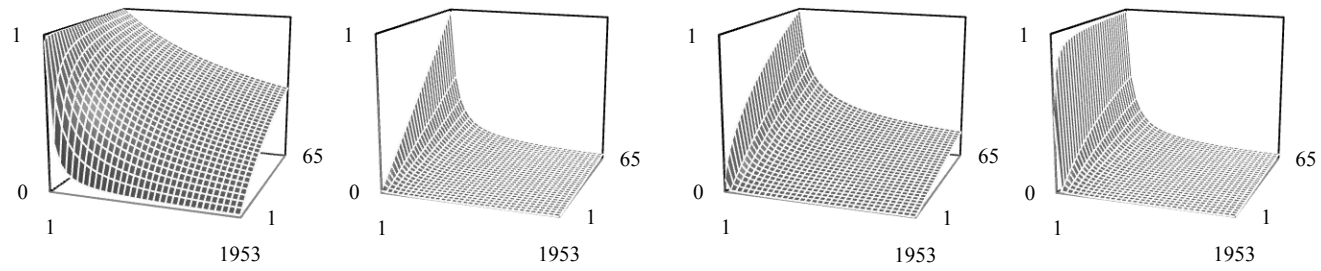
We cluster realistic faults according to their execution characteristics to find potential fault patterns. For each fault pattern, we calculate the mean execution characteristics, count the frequency of faults for that pattern, and generate a central point representing that pattern. Using spline method, we work out a 3D surface through all the central points, and use it to build our statistical fault localization technique.

We use a common data set, Siemens suite, to conduct cross validation test to evaluate the effectiveness of our model and compare the result of our technique with four existing representative techniques, Tarantula [10], Jaccard [2], Ochiai [2], and SBI [15]. The empirical results show that our technique is more effective than the other techniques, on the Siemens suite.

The rest of the paper is organized as follows. Section II gives related work. Section III introduce statistical fault localization framework, then use a 3D surface representation to visualize the suspiciousness function used in previous such techniques. Section IV discuss the properties of previous techniques, motivates our work, and elaborate on

*This research is supported by the National High Technology Research and Development Program of China (project no. 2007AA01Z145).

978-0-7695-4043-6/10 $26.00 © 2010 IEEE
DOI 10.1109/ICCRD.2010.165



(a) $R_{Tarantula}$ (s_i) for Tarantula (b) $R_{Jaccard}$ (s_i) for Jaccard (c) R_{Ochiai} (s_i) for Ochiai (d) R_{SBI} (s_i) for SBI

Figure 1. 3D surface representation of suspiciousness functions.

our model. Section V gives empirical evaluation. Section VI concludes this paper.

II. Related Work

Comparing program execution information to facilitate fault localization is a frequently used strategy of fault localization. For example, Delta Debugging [3][4] considers program execution as a sequence of states. It compares passed execution and failed execution to shorten the suspicious region, and traces back to locate the failure cause.

Tarantula [10] counts the chance of a statement being exercised in failed executions and that in passed executions. It then uses such two chances to assess the suspiciousness of that statement of being a fault. Jaccard [2], Ochiai [2], and SBI [15] are similar statement-level such techniques.

Statistical fault localization can be also conducted at predicate-level. Such work includes CBI [11] and SOBER [12]. They accordingly evaluate the suspiciousness of a predicate statement being related to fault. A metrics t-score is proposed in a previous study Nearest Neighbor [13], to evaluate the effectiveness of these techniques.

There are also other kind of fault localization techniques. Value Replacement [8] varies each variable to all possible values, to find those ones which trigger may change a failed execution to a passed execution, and use such method to locate fault around the key variable found. It is also very effective. However, its efficiency is relatively lower than the other statistical techniques. CP [17] makes use of control-flow information to enhance fault localization. Comparing with Tarantula, Jaccard, Ochiai, and SBI, it involves additional input information.

Instead of locating single fault in programs, Jones et al. [9] investigate locating multiple faults in parallel. Different from those techniques that focus on more precise fault position prediction, Wang et al. [14] alter the quality of executions in view of coverage, to improve the effectiveness of fault localization.

III. Surface Representation

In this section, we first recall current statistical fault localization framework and existing popular statistical fault localization techniques. We then employ a 3D surface representation to visualize the core suspiciousness function of those techniques.

A. Statistical Fault Localization Framework

Previous statistical fault localization techniques, such as Tarantula [10], Ochiai [2], Jaccard [2], CBI [11], SOBER [12], SBI [15], use a suspiciousness function to evaluate the suspiciousness of a statement being related to fault. They then rank all suspicious program elements according to their thus calculated suspiciousness value, and generate a ranked list of program elements. Such a list is helpful for programmer to debug [13]. Programmers check statements in the list, from the most suspicious (top of the list) to the least suspicious (tail of the list), to locate a fault. And the position of the faulty statement found in the list is used as a measurement of the effectiveness of a statistical fault localization technique.

In the next section, we use four previous representative statistical fault localization techniques to illustrate previous studies in above framework.

B. Statistical Fault Localization Techniques

Given a set of executions (e.g., m failed executions and m' passed executions), Tarantula's suspiciousness function $R_{Tarantula}(s_i)$ for statement s_i is shown as below. The terms $failed(s_i)$ and $passed(s_i)$ are respectively the number of failed and passed executions that exercise statement s_i.

$$R_{Tarantula}\ (s_i) = \frac{failed(s_i)/m}{failed(s_i)/m + \ passed(s_i)/m'} \quad (1)$$

For comparison purpose, we also list out the suspiciousness functions of Jaccard, Ochiai, and SBI, as follows.

$$R_{Jaccard}\ (s_i) = \frac{failed(s_i)}{m + passed(s_i)} \quad (2)$$

$$R_{Ochiai}\ (s_i) = \frac{failed(s_i)}{\sqrt{m \times (passed(s_i) + failed(s_i))}} \quad (3)$$

$$R_{SBI}\ (s_i) = \frac{failed(s_i)}{failed(s_i) + passed(s_i)} \quad (4)$$

Note that the difference among these four techniques is their suspiciousness functions, and the ranking order of two statements s_{i1} and s_{i2} is determined by the suspiciousness function used in each technique. Suppose we use technique $T1$ and get the ranking order $R_{T1}(s_{i1}) > R_{T1}(s_{i2})$, it means that

the statement s_{i1} is deemed to be more suspicious than the statement s_{i2}, when using technique *TI* to evaluate.

In next section, we show how we visualize these suspiciousness functions to give an intuitive understanding.

C. Surface Representation of Suspiciousness Function

We notice that when given a set of executions (e.g., *m* failed executions and *m'* passed executions), the value of the suspiciousness for statement s_i is function (e.g., $R_{Tarantula}(s_i)$) of variables $failed(s_i)$ and $passed(s_i)$, which are respectively the number of failed and passed executions that exercise statement s_i. Therefore, we use a 3D surface to represent the suspiciousness function. The depth, horizontal, and vertical coordinates stand for $failed(s_i)$, $passed(s_i)$, and $R_{Tx}(s_i)$, respectively. Here $failed(s_i) \in \{1, m\}$, $passed(s_i) \in \{1, m'\}$, $R_{Tx}(s_i) \in \{0, 1\}$, and $T_x = T_{Tarantula}$, $T_{Jaccard}$, T_{Ochiai}, or T_{SBI}.

We plot suspiciousness functions of Tarantula, Jaccard, Ochiai, and SBI in Fig. 1(a), Fig. 1(b), Fig. 1(c), and Fig. 1(d), respectively. In each plot, variable $failed(s_i)$ varies in range [1, 65], which means that *m* is specified as 65; variable $passed(s_i)$ varies in range [1, 1953], which means that *m'* is specified as 1953. We choose these magic numbers to demonstrate because they are calculated as the mean number of passed executions and mean number of failed executions of the well-known common data set, Siemens suite. In each plot, the horizontal axis shows the value of variable $passed(s_i)$, the depth axis shows the value of variable $failed(s_i)$, and the vertical axis shows the value of suspiciousness function. The higher the vertical coordinates, the higher suspiciousness function results are.

IV. Our Fault Localizatoin Model

In this section, we first investigate the properties of previous fault localization techniques, using the 3D surface representation of suspiciousness functions. After that and motivated by properties of previous techniques, we propose our model and elaborate on our fault localization technique.

A. Properties of Previous Techniques

From Fig. 1, we notice that in each plot, the function value increases with the increasing of variable $failed(s_i)$, when variable $passed(s_i)$ is fixed. On the other hand, the function value decrease with the increasing of variable $passed(s_i)$, when variable $failed(s_i)$ is fixed. Formally, our observation is that all of these four suspiciousness functions satisfy the following two properties. The straightforward proof is not given in this paper.

[Property 1: Monotonically increasing] The value of suspiciousness function $R_{Tx}(s_i)$ is a monotonically increasing function of variable $failed(s_i)$, i.e., $\forall s_{i1}, s_{i2}, failed(s_{i1}) \geq failed(s_{i2}) \wedge passed(s_{i1}) = passed(s_{i2}) \Longrightarrow R_{Tx}(s_{i1}) \geq R_{Tx}(s_{i2})$.

[Property 2: Monotonically decreasing] The value of suspiciousness function $R_{Tx}(s_i)$ is a monotonically decreasing function of variable $passed(s_i)$, i.e., $\forall s_{i1}, s_{i2}, failed(s_{i1}) = failed(s_{i2}) \wedge passed(s_{i1}) \geq passed(s_{i2}) \Longrightarrow R_{Tx}(s_{i1}) \leq R_{Tx}(s_{i2})$.

Property 1 is explained as follows. For two statements s_{i1} and s_{i2} that are exercised in identical number of the passed executions, we turn to the failed executions to distinguish their suspiciousness, and regard the one exercised in more failed executions as more suspicious. That is because that the more a statement is exercised by failed executions, the more probable it may be the cause of those failed executions.

Similarly, property 2 is explained as follows. For two statements s_{i1} and s_{i2} that are exercised in identical number of the failed executions, we turn to the passed executions to distinguish their suspiciousness, and regard the one exercised in more passed executions as less suspicious. That is because that the more a statement is exercised in passed executions, the more safe (less suspicious) it may not cause a failed execution.

These two properties seem intuitive for a statistical fault localization technique. However, strictly following them do not result in best effectiveness on realistic programs and faults. We will give a motivation in next section.

B. Fault Pattern in Realistic Programs

In realistic programs, faults do not evenly appear everywhere. On the contrary, realistic faults may highly related to special program structures or seldom appear in specific procedures. As a result, if we blindly assess the suspiciousness of program elements according to their corresponding $failed(s_i)$ value and $passed(s_i)$ value, the distribution information of realistic faults are missing.

[Case 1: Main module and function entrance] In realistic programs, many faults are related to assignment statements in main modules and function entrance. According to previous studies [6], among various C programs, 22.8% faults are of such a type. Since the main module or function entrance of a program are exercised by almost all the executions, both $failed(s_i)$ and $passed(s_i)$ have greatest values. Program elements in such cases have relatively low suspiciousness value (calculated as about 0.5 by Tarantula, less than 0.2 by Ochiai, and close to 0 by Jaccard and SBI), however, they should be given much more attention. To locate such a majority of faults effectively, the simplest strategy is to increase the value of suspiciousness function in the corresponding range of the 3D surface.

[Case 2: Function exit and return statement] In realistic faults, only limited faults exist in function exit and return statements. According to previous studies [6], among variant C programs, only about 7.8% faults are of such a type. Since an incorrect function exit or wrong return statement may immediately cause a failure, $failed(s_i)$ have greatest value, but $passed(s_i)$ is very low. Program elements in such cases have relatively high suspiciousness value (calculated as greater than 0.8 by each technique), however, their intrinsic suspiciousness are not that high. To save effort on such a minority of faults, the simplest strategy is to decrease the value of suspiciousness function in the corresponding range of the 3D surface.

We next elaborate on how we recognize fault patterns and construct an effective 3D surface.

C. Our Model

In our model, we apply the previous framework and construct a 3D surface for the suspiciousness function to formulate our technique.

The construction of the 3D surface consists of three steps. To determine the shape of 3D surface and make an effective suspiciousness function for realistic programs, we learn fault patterns from realistic faults first. We cluster the faulty statements of each faulty version in the training set, to cluster the faults into various fault patterns. In each of the clusters, we calculate a central point to stand for the characteristics of that fault pattern. After that, we use spline method to construct a 3D surface through the central points.

[Step 1: Cluster] In this step, we collect tuple of $<failed(s_i), passed(s_i), R_{Tx}(s_i)>$ for faulty statement in each faulty version. In subsection B, we observe that most fault patterns can be recognized by characteristics on the values of $failed(s_i)$ and $passed(s_i)$. Therefore, we deem each value pair of $<failed(s_i), passed(s_i)>$ as a 2D point and employ cluster techniques to cluster the tuples by such 2D points. After such a clustering, we deem that each cluster maps to a different fault pattern.

[Step 2: Centralize] In this step, we calculate a central point for each cluster. The values of coordinates $\overline{failed(s_i)}$ and $\overline{passed(s_i)}$ of the central point capture the execution characteristics of that fault pattern. The value of coordinate $\overline{R_{Tx}(s_i)}$ captures the suspiciousness value designed to given to statements in that fault pattern. For each cluster, they are calculated using the following formulas, where $s_1, s_2, \ldots, s_n$ means the faulty statements clustered in that class. In these formulas, the value of $\overline{failed(s_i)}$ and $\overline{passed(s_i)}$ are calculated using the mean of corresponding values.

$$\overline{failed(s_i)} = \frac{1}{n}\sum_{j=1}^{n}\left[failed(s_j)\right] \quad (5)$$

$$\overline{passed(s_i)} = \frac{1}{n}\sum_{j=1}^{n}\left[passed(s_j)\right] \quad (6)$$

$$\overline{R_{Tx}(s_i)} = n \times \frac{1}{n}\sum_{j=1}\left[R_{Tx}(s_j)\right] = \sum_{j=1}\left[R_{Tx}(s_j)\right] \quad (7)$$

The value of $\overline{R_{Tx}(s_i)}$ is calculated as n (cluster size) times the mean value of corresponding values. So the larger the scale of the cluster is, the more suspiciousness will be given to it. On the contrary, the smaller the scale of the cluster is, the less effort will be put on it. For clusters of identical scale, the one having a larger mean suspiciousness value will be given higher suspiciousness values.

[Step 3: Spline] In this step, we use spline method to construct a 3D surface through the generated central points. We use such a 3D surface to build suspiciousness function for our statistical fault localization technique.

Note that we do not limit the use of cluster and spline method, and in fact, any cluster and spline method can be used in our model. In next section, we will introduce the model settings and the experiment results of our fault localization technique.

TABLE I. SUBJECT PROGRAMS – SIEMENS SUITE [5].

Programs	*Number of statements*	*Number of faulty versions*	*Percentage of failed executions*
print_tokens	341 – 342	7	1.7%
print_tokens2	350 – 354	10	5.4%
replace	508 – 515	31	2.0%
schedule	291 – 294	5	3.2%
scheudule2	261 – 263	9	1.0%
tcas	133 – 137	41	2.4%
tot_info	272 – 274	23	5.6%

V. EMPIRICAL EVALUATION

In this section, we first introduce the experiment settings, and then give the empirical results and related discussions.

A. Experiment Settings

In this experiment, we compare the effectiveness of our technique with four previous representative techniques. They are Tarantula [10], Jaccard [2], Ochiai [2], and SBI [15]. These four techniques are all statement-level fault localization techniques. For space reason, we do not adapt our technique to predicate-level to compare with other predicate-level techniques.

The subject programs used in our experiment is the widely used common data set, Siemens suite [5]. Siemens suite was developed to support debugging and testing studies and was frequently used in previous studies [4][7][10][11][12][13]. It includes seven programs; each program has several faulty versions. In each faulty version, there is a fault. A generated set of executions is attached with each program. Table I shows the statistics of Siemens suite. Let us take the first row as example. It says that the program "print_tokens" has seven faulty versions, each of which has one fault. These faulty versions have 341 to 342 lines of code. Among the given set of executions, about 1.7% of them are failed executions. Note that some faulty versions are not feasible [12], so we exclude them. Some other faulty versions are not compatible with the instrumentation tool gcov [10], so we also exclude them.

For each faulty version, we follow previous studies [12] to collect execution information for the given set of executions. We randomly separate the 126 faulty versions into two parts, that is, training set and data set. After that, we apply Tarantula, Jaccard, Ochiai, SBI, and our technique to get the suspiciousness value for each statement and sort them to form a ranked list. We check along the generated list from top (most suspicious statement) to tail (least suspicious statement) to search for fault, and use the position of fault in the list as the effectiveness for comparison. For example, in the generated list of 341 statements, suppose the faulty statement is finally ranked at the 23rd position in the generated list. Therefore, the effort to locate this fault is calculated as 23 / 341 × 100% ≈ 6.7%, which means we need to examine 6.7% statements in the resultant list to reach the fault. We further use 1 – 6.7% = 93.3% to evaluate the

effectiveness; the greater, the better. Formally, the effectiveness is calculated as follows.

$$\left(1 - \frac{\text{position of faulty statement in list}}{\text{number of statements in list}}\right) \times 100\% \quad (8)$$

B. Model Settings

In the experiment, we use the original k-nearest neighbor method [16] for clustering in our model. The parameter K is chosen as 4; Euclidean distance is used. Since the number of passed execution and that of failed execution may not be equal. We first normalize *failed*(s_i) and *passed*(s_i) to *failed*(s_i)/*m* and *passed*(s_i)/*m'* respectively, before applying the Euclidean distance. The B-spline method [16] is used to construct surface from 3D points.

C. Experiment Results

Fig. 2 shows the box plot of effectiveness comparison results for the five techniques. In this figure, effectiveness statistics for each technique on the 126 Siemens faulty versions is shown as a box. The top and bottom of the box respectively mean the 75% and 25% percentiles in the set of 126 effectiveness values. The band near the middle means the 50% percentile (median). The top and bottom ends of the whisker mean the 90% and 10% percentiles, respectively.

From the results, we observe that the effectiveness of our technique over Siemens suite is better than those of the other techniques. For example, the median effectiveness of our technique is 92.6%, while the median effectiveness of Tarantula, Jaccard, Ochiai, and SBI are 84.5%, 84.6%, 84.7%, and 84.3%, respectively.

D. Discussions

Though our technique seems to show some advantage to the other techniques, we also find that the standard deviation for Tarantula, Jaccard, Ochiai, SBI, and our techniques are 14.4%, 14.3%, 14.3%, 14.3%, and 17.5%, respectively. It means that our technique has relatively worse stability.

Faults in Siemens suite are seeded manually. Other realistic programs may be used to evaluate our technique to further strengthen the validity of experiment.

VI. Conclusion

Fault localization is a time-consuming task in software engineering. By analyzing program execution information, previous statistical fault localization techniques make use of suspiciousness function to assess the suspiciousness of each program element of being fault. In this paper, we use a 3D surface representation to investigate the property of previous techniques. We cluster faults from execution characteristics to find potential fault patterns, and optimize suspiciousness function by constructing a 3D surface for representative fault patterns. The empirical results show that our method outperforms previous representative techniques on the common data set, Siemens suite. Future work may include developing the technique to fit fault localization in concurrent programs.

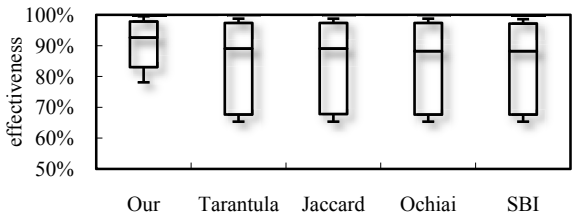


Figure 2. Box plot of effectiveness comparison.

References

[1] P. Arumuga Nainar, T. Chen, J. Rosin, and B. Liblit, "Statistical debugging using compound Boolean predicates", in *ISSTA '07*, ACM Press, 2007, pp. 5–15.

[2] R. Abreu, P. Zoeteweij, and A. J. C. van Gemund, "On the accuracy of spectrum-based fault localization", in *TAICPART-MUTATION '07*, IEEE Computer Society, 2007, pp. 89–98.

[3] L. C. Briand, Y. Labiche, and X. Liu, "Using machine learning to support debugging with Tarantula", in *ISSRE '07*, IEEE Computer Society, 2003, pp. 137–146.

[4] H. Cleve and A. Zeller. "Locating causes of program failures", in *ICSE '05*, ACM Press, 2005, pp. 342–351.

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

[6] J. A. Durães and H. S. Madeira, "Emulation of software faults: a field data study and a practical approach", *IEEE TSE*, 32(11):849–867, 2006.

[7] S. Elbaum, G. Rothermel, S. Kanduri, and A. Malishevsky, "Selecting a cost-effective test case prioritization technique", *Software Quality Journal*, 12(3): 185–210, 2004.

[8] D. Jeffrey, N. Gupta, and R. Gupta, "Fault localization using value replacement", in *ISSTA '08*, ACM Press, 2008, pp. 167–178.

[9] J. A. Jones, J. F. Bowring, and M. J. Harrold, "Debugging in parallel", in *ISSTA '07*, ACM Press, 2007, pp. 16–26.

[10] J. A. Jones and M. J. Harrold, "Empirical evaluation of the Tarantula automatic fault-localization technique", in *ASE '05*, ACM Press, 2005, pp. 273–282.

[11] B. Liblit, M. Naik, A. X. Zheng, A. Aiken, and M. I. Jordan, "Scalable statistical bug isolation", in *PLDI '05*, ACM Press, 2005, pp. 15–26.

[12] C. Liu, L. Fei, X. Yan, J. Han, and S. P. Midkiff, "Statistical debugging: a hypothesis testing-based approach", *IEEE TSE*, 32(10):831–848, 2006.

[13] M. Renieris and S. P. Reiss, "Fault localization with nearest neighbor queries", in *ASE '03*, IEEE Computer Society, 2003, pp. 30–39.

[14] X. Wang, S. C. Cheung, W. K. Chan, and Z. Zhang, "Taming coincidental correctness: coverage refinement with context to improve fault localization", in *ICSE '09*, IEEE Computer Society Press, 2009, pp. 45–55.

[15] Y. Yu, J. A. Jones, and M. J. Harrold, "An empirical study of the effects of test-suite reduction on fault localization", in *ICSE '08*, ACM Press, 2008, pp. 201–210.

[16] D. G. Zill and M. R. Cullen, *Advanced Engineering Mathematics*, Jones and Bartlett Publishers, 2006.

[17] Z. Zhang, W. K. Chan, T. H. Tse, B. Jiang, and X. Wang, "Capturing propagation of infected program states", in *FSE '09*, ACM Press, 2009, pp. 43–52.