ICIC Express Letters
Part B: Applications
Volume **7**, Number **12**, December **2016**

ICIC International ©2016 ISSN 2185-2766
pp. **2519–2526**

COMPREHENSIVE STATEMENT RANKING ALGORITHM BASED ON SUSPICIOUSNESS AND DEPENDENCE FOR FAULT LOCALIZATION IN SOFTWARE SYSTEM

Wanchang Jiang[1,2], Jiadong Ren[1] and Yuan Huang[1]

[1]College of Information Science and Engineering
Yanshan University
No. 438, Hebei Avenue, Qinhuangdao 066004, P. R. China
jwchang84@163.com; jdren@ysu.edu.cn; 757918272@qq.com

[2]School of Information Engineering
Northeast Dianli University
No. 169, Changchun Road, Jilin 132012, P. R. China

Received June 2016; accepted September 2016

Abstract. *Fault localization of statement granularity is important for software testing. In this paper, in order to solve the ineffectiveness of failed execution spectrum-based suspiciousness metric, both failed and successful non-execution spectra are considered as decisive factors to design a new suspiciousness computation method (short as ENS), and then the contribution of each statement to the fault can be determined. With the proposed concept of execution trace self-information, each expression in ENS metric is weighted respectively to reflect the importance of each decisive factor, and then a weighted suspiciousness computation method ENSω is proposed. With the dynamic dependence information of failed executions, a comprehensive statement ranking algorithm is designed on the basis of suspiciousness metric for fault localization. Experiments are carried out on the typical program in Siemens Suite for verifying better effectiveness of the methods ENS and ENSω for fault localization than other metrics, especially ENSω. It is shown that our comprehensive statement ranking algorithm has better performance, and fewer statements need to be inspected until fault is found.*

Keywords: Software fault localization, Comprehensive statement ranking, Suspiciousness metric, Dependence information

1. Introduction.

Since the human factors are involved in software system development process, it is necessary to improve and confirm software reliability. To improve software stability and robustness of function granularity, a method is designed to measure the importance of functions in software network [1]. Fault of statement granularity can be located with module testing, and every unit in system could realize its expected function. Because of the complexity of software system, software testing and debugging is high in time-consumption with test cases [2]. Test case prioritization and selection techniques are designed to improve the efficiency of software testing, which reduces testing cost to a large extent [3, 4].

To effectively solve the problem with the given test cases, fault localization techniques are proposed, which can be realized based on program spectra. Program spectra including statement spectra are designed to obtain the statistical information of execution of program. To assist localizing the fault in a fault program, suspiciousness metrics are designed by using some of spectra to compute suspiciousness value for each statement to be the fault. Since only a few test cases are failed ones, no failed execution makes metrics of WONG1 [5], Tarantula [6], Zoltar [7] and Ochiai [8] ineffective. Furthermore, the performance of metrics of Sokal, Hamann [6] and WONG2 [5] need to be improved, whose decisive spectra have the same effect on the suspiciousness.

The suspiciousness computation metric is not good in locating the fault for some cases wherein a fault may be missing partial code or in a condition statement. The fault may change the value of parameter in the statement, or the execution trace. Without coverage information, it is difficult to locate fault only by using the suspiciousness score of each statement. As a result, only with the program spectra, the performance of locating the fault statement is not stable.

Therefore, a new suspiciousness computation method ENS is designed based on failed and successful non-execution spectra. By using the proposed execution trace self-information, a weighted suspiciousness computation method ENSω is designed based on ENS. Furthermore, with the dynamic dependence information of failed executions, related statements in execution can be considered together to gather information about fault, and then a comprehensive statement ranking algorithm is presented to further improve the performance of suspiciousness-based fault localization with given test cases.

The organization of this paper is as follows. Section 2 discusses the preliminaries. Section 3 introduces the suspiciousness computation method ENS and the weighted suspiciousness computation method ENSω. Section 4 describes the comprehensive statement ranking algorithm for fault localization. In Sections 5, the experiments are conducted on the typical program in Siemens Suite. Section 6 concludes the work and discusses possible directions for future work.

2. Preliminaries.

In this section, concepts of fault program, test suite, program spectra and execution trace self-information are described.

Definition 2.1. *For a fault program which contains one fault, candidate statement set containing the fault is denoted as* $\{S_1, \cdots, S_i, \cdots, S_N\}$, *where* S_i *is the ith statement and* N *is the number of statements.*

Definition 2.2. *A fault program will be executed with a test suite of test cases, which is denoted as* $\{T_1, \cdots, T_j, \cdots, T_M\}$, *where* T_j *is a test case and* M *is the number of test cases. According to the execution result, test cases are divided into passed and failed ones.* T_j *is a failed test case when the execution is a failed one; otherwise* T_j *is a passed test case when the execution is a successful one.*

Definition 2.3. *Program spectra of* $a_{ef}(S_i)$, $a_{ep}(S_i)$, $a_{nf}(S_i)$ *and* $a_{np}(S_i)$ *are defined to extract information from execution traces of test running, short as* a_{ef}, a_{ep}, a_{nf} *and* a_{np}. *Failed execution spectrum* a_{ef} *denotes the number of failed executions covering the statement* S_i. *Failed non-execution spectrum* a_{nf} *denotes the number of failed executions not covering the statement* S_i. *And successful execution spectrum* a_{ep} *and successful non-execution spectrum* a_{np} *respectively denote the number of successful executions covering and not covering statement* S_i.

Definition 2.4. *With execution traces, the probability of failed execution covering* S_i *is used to compute failed execution trace self-information* $h_{ef}(S_i)$ *which is proposed to gather information that statement* S_i *is covered by the failed execution.*

$$h_{ef}(S_i) = -P(S_iR)\log(P(S_iR)) \tag{1}$$

Failed non-execution trace self-information $h_{nf}(S_i)$, *successful execution trace self-information* $h_{ep}(S_i)$ *and successful non-execution trace self-information* $h_{np}(S_i)$ *can be defined similarly.*

3. Suspiciousness Computation Metrics ENS and ENSω.

A new suspiciousness metric ENS is proposed on the basis of failed execution spectrum and successful non-execution spectrum. Furthermore, two other spectra are also utilized for constructing suspiciousness computation formula. By using the proposed execution trace self-information, a weighted suspiciousness metric ENSω is designed based on ENS.

3.1. **Suspiciousness metric ENS.** When a statement is executed only by the failed test cases, the statement has a high likelihood of causing fault. In addition, if the statement is not covered by most successful executions, the statement may be related to fault and provide information about the fault. Thus, both failed execution spectrum and successful non-execution spectrum are considered as the decisive factors of computing the suspiciousness, and a_{ef}-based expression and a_{np}-based expression are constructed respectively by using some spectra. As a result, a novel suspiciousness computation method ENS is designed, and the formula is as follows.

$$ENS(S_i) = \frac{a_{ef}}{a_{ef} + a_{ep} + a_{nf}} + \frac{a_{np}}{a_{np} + a_{ep} + a_{nf}} \tag{2}$$

The event of one successful execution covering a statement decreases the possibility of the statement to be the fault, and the event of one failed execution not covering a statement decreases the possibility. So, in the construction process of a_{ef}-based expression and a_{np}-based expression, the factors of failed non-execution and successful execution spectra are also considered to get more comprehensive information about fault. Both a_{ep} and a_{nf} are used as inverse factors to reflect the influence of a_{ep} and a_{nf} on the fault. a_{ef} is included in the denominator to reduce the importance of a_{ef}-based expression. As a result, the sum of a_{ef}, a_{ep} and a_{nf} is considered as the denominator of a_{ef}-based expression. In a similar way, the sum of a_{np}, a_{ep} and a_{nf} is used to construct a_{np}-based expression to balance the influence of a_{np}.

3.2. **Weighted suspiciousness computation metric ENSω.** To reflect the influence of each expression of ENS formula upon suspiciousness, a weighted suspiciousness computation method ENSω is proposed based on method ENS by using execution trace self-information.

With the proposed failed execution trace self-information $h_{ef}(S_i)$, failed non-execution trace self-information $h_{nf}(S_i)$ and successful execution trace self-information $h_{ep}(S_i)$ and successful non-execution trace self-information $h_{ep}(S_i)$, two weights, abbreviated as ω_{ef} and ω_{np} respectively, are presented for the a_{ef}-based expression and a_{np}-based expression in ENS, which are shown in the following formulas.

$$\omega_{ef} = \frac{h_{ef}(S_i)}{h_{ef}(S_i) + h_{ep}(S_i) + h_{nf}(S_i)} \tag{3}$$

$$\omega_{np} = \frac{h_{np}(S_i)}{h_{np}(S_i) + h_{ep}(S_i) + h_{nf}(S_i)} \tag{4}$$

$h_{ef}(S_i)$, $h_{nf}(S_i)$ and $h_{ep}(S_i)$ are considered to design ω_{ef}, and the structure form is consistent with that of the expression to be weighted. Therefore, it is not necessary to consider the symbol factor of $h_{ef}(S_i)$, $h_{nf}(S_i)$ and $h_{ep}(S_i)$, where $h_{ef}(S_i)$ is proportional to ω_{ef}, and $h_{ep}(S_i)$ and $h_{nf}(S_i)$ are inversely proportional to ω_{ef}. To balance the value of weights and give different weights to each part to be weighted, $h_{ef}(S_i)$ is put as a component of the denominator.

Each expression of ENS is considered as a whole and weighted by ω_{ef} and ω_{np} respectively, and then a weighted suspiciousness metric ENSω is proposed.

$$\text{ENS}\omega(S_i) = \omega_{ef} \cdot \frac{a_{ef}}{a_{ef} + a_{ep} + a_{nf}} + \omega_{np} \cdot \frac{a_{np}}{a_{np} + a_{ep} + a_{nf}} \tag{5}$$

The different contribution of each expression to the suspiciousness is reflected. A high value of $h_{ef}(S_i)$ means a high value of ω_{ef}, which implies a high probability that S_i causes the fault. Similarly, a high value of $h_{np}(S_i)$ indicates a high probability that S_i causes the fault.

4. **The Comprehensive Statement Ranking Algorithm for Fault Localization.**

A comprehensive statement ranking algorithm based on suspiciousness and dependence information is designed, which applies suspiciousness result and dynamic dependence information of failed execution to ranking statements of a given fault program.

For a fault program, work of four phases should be completed in the algorithm to obtain comprehensive statement ranking. First of all, the program is compiled and executed with

Algorithm 1: The comprehensive statement ranking algorithm

```
Input: fault versions of program, test suite $\{T_j\}$
Output: comprehensive statement sequence for each version with each metric
1. For each fault version $V_k$
2.   Compile fault program
3.   Gather static dependence information
4. End for
5. For each fault version
6.   For each test case $T_j$
7.     Execute fault program with test case $T_j$
8.     Collect tracing information of the execution
9.     Compare actual result with expected result, output $r_j$
10.  End For
11.End For
12.For each fault version
13.  For each failed test case $T_j$
14.    Extract execution trace from tracing information
15.    Gather dynamic control dependence information $\{S_{j_i} \to S_{j_k}\}$
16.    Gather dynamic data dependence information $\{S_{j_p} \to S_{j_q}\}$
17.  End For
18.End For
19.For each fault version
20.  For each metric
21.    For each statement $S_i$
22.      Compute $a_{ef}(S_i)$, $a_{ep}(S_i)$, $a_{nf}(S_i)$ and $a_{np}(S_i)$ by using execution traces
23.      Compute $h_{ef}(S_i)$, $h_{ep}(S_i)$, $h_{nf}(S_i)$ and $h_{np}(S_i)$ by using execution traces
24.      Compute suspiciousness of statement $S_i$ by the metric
25.    End For
26.    Rank statements on basis of suspiciousness result
27.    Obtain sequence $\{S_{i_1}, S_{i_1} \cdots S_{i_N}\}$
28.  End For
29.End For
30.For each fault version
31.  If {failed test cases} $\neq \varnothing$
32.    For each metric
33.      For each statement $S_{i_k}$ in $\{S_{i_1}, S_{i_1} \cdots S_{i_N}\}$
34.        Get its control and data dependence statements
35.        Rank these statements with suspiciousness
36.        Insert its dependence statements after the statement
37.      End For
38.      Output the comprehensive statement sequence
39.    End For
40.  End If
41.End For
```

test cases, and execution traces and results are recorded. In addition, with tracing information of failed execution, the control and data dependence statements are gathered, and the related statements in execution are obtained. Then, the suspiciousness of statements can be computed by using the suspiciousness metrics. At last, on the basis of suspiciousness result, with the control and data dependence information, the comprehensive statement ranking is obtained.

The comprehensive statement ranking algorithm is presented as in Algorithm 1.

With our proposed suspiciousness metrics or other metrics, all statements are ranked as the sequence of statements $\{S_{i_1}, S_{i_2} \cdots S_{i_N}\}$ based on suspiciousness. For the statement S_{i_k}, its control and data dependence statements of failed execution are ranked as the subsequence $\{S_{i_k 1} \cdots\}$ with suspiciousness. Then the subsequence is added to the comprehensive ranking sequence of statements $\{S_{i_1}, \{S_{i_1 1} \cdots\} \cdots S_{i_k}, \{S_{i_k 1} \cdots\} \cdots S_{i_N}\}$. If one statement has emerged in the sequence, it is unnecessary to add the statement. As a result, the comprehensive ranking sequence $\{S_{i_1}, \{S_{i_1 1} \cdots\} \cdots S_{i_k}, \{S_{i_k 1} \cdots\} \cdots S_{i_N}, \{S_{i_N 1} \cdots\}\}$ is finally obtained. Then programmer can inspect statements according to the order of each statement in the sequence until the fault is located.

As shown in 34-36 lines of algorithm, the suspiciousness-based statement ranking result is improved by using ranked control and data dependence statements. Besides the ranked control and data dependence strategy, other four different strategies of using dependence information to obtain comprehensive statement can be used, which are the data dependence strategy, the control and data dependence strategy, the data and control dependence strategy and the ranked data dependence strategy. The performance of each strategy will be discussed in the experiment.

That there is none of failed test cases means there is none dependence information of failed execution. Therefore, the suspiciousness-based statement ranking sequence is considered as the final result.

5. **Experiment.**

The performance of our proposed metrics for fault localization is compared with that of a_{ef}-based metric Tarantula (TA), and spectra-based metrics Hamann (HAN) and WONG2. And the performance of comprehensive ranking for fault localization with different strategies is discussed.

5.1. **Experiment setup.**

Experiments are conducted on the typical program of "tcas" in Siemens Suite. Different faults were seeded as realistic as possible, and 41 fault versions of program "tcas" are provided. And 35 versions are selected in the experiment. A large test pool containing possible test cases was created as "Universe" suite. To reduce the time-consumption of fault localization, test suites of two types "bigrand" and "bigcov" are utilized. Test suite of "bigcov" type was generated to achieve branch coverage. With the same size of "bigcov", test suite of "bigrand" was generated randomly. To avoid the particularity of one test suite, four test suites of each type are used to investigate the average performance. Experiment environment is Fedora Core System, and the comprehensive statement ranking algorithm is realized by Java programming language, which refers to the software infrastructure of WET [9].

5.2. **Experiment results and analysis.**

Based on suspiciousness results with suites of "bigrand" type, the average ranking of fault of each fault version with each metric is obtained, which is shown in Figure 1. From 41 faulty versions, 35 are chosen and used. The other versions like versions 10 and 11 are abandoned.

A stable average ranking of fault of "tcas" with "bigrand" suites is obtained by using our proposed metrics. Our metrics, especially ENSω, outperform metrics of TA, HAN and WONG2. ENS gains an average increase of 10.4%, 0.5% and 0.5% respectively, and ENSω gains an average increase of 14.8%, 4.9% and 4.9% respectively. Furthermore, ENSω performs better than ENS for many versions, such as versions 2, 4, 6, 9. Metric

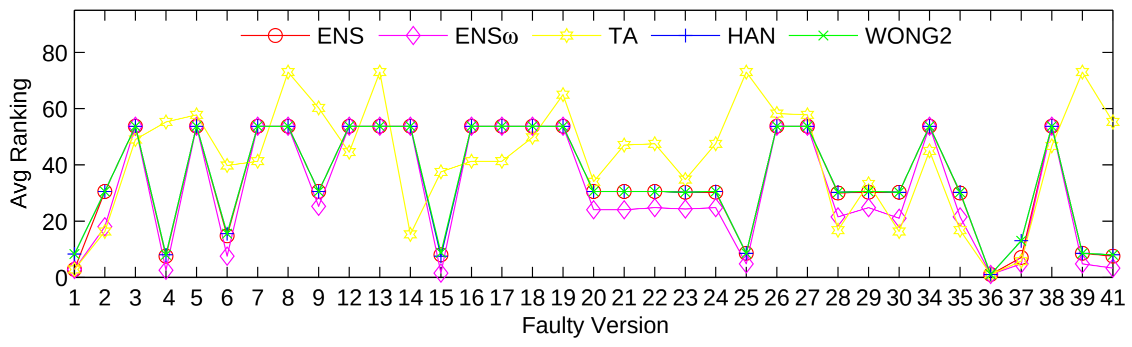

FIGURE 1. The average ranking of fault with "bigrand" suites

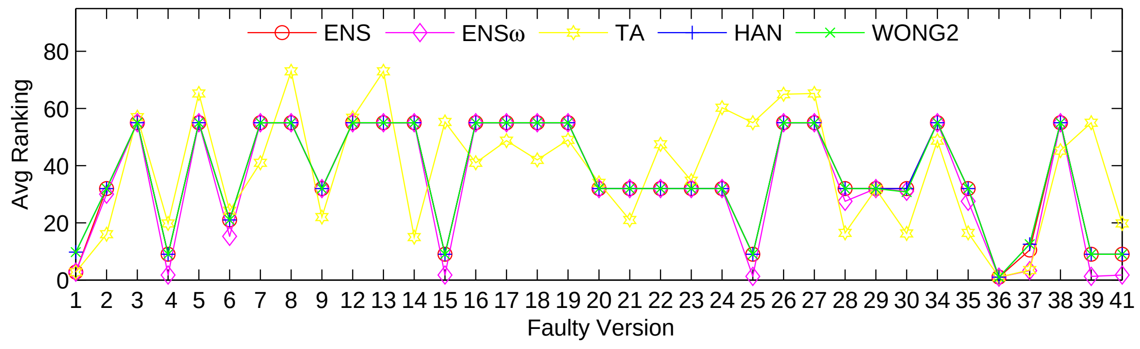

FIGURE 2. The average ranking of fault with "bigcov" suites

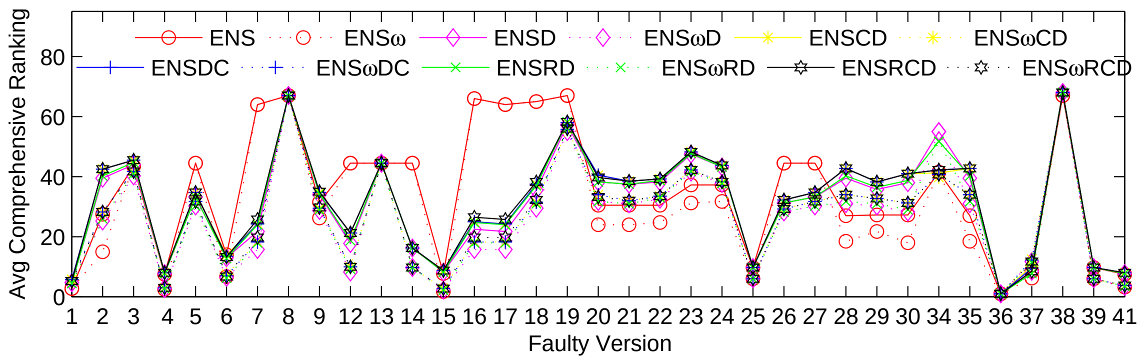

FIGURE 3. The average comprehensive ranking of fault with "bigrand" suites

of TA is obviously ineffective for some versions, such as versions 8, 13, 19 and 39. In contrast, our metrics can even work in this case, and ineffectiveness of a_{ef}-based metric is solved.

With test suites of "bigcov" type, the average ranking of fault of each version with each metric is shown in Figure 2.

Even with "bigcov" suites, a stable performance can be obtained by using our proposed metrics. ENS gains an average increase of 1.3%, and up to 3.2% in specific case over the three metrics. ENSω increases the average ranking about 5.6%, 2.8% and 2.8% on average. The performance of TA is not stable, and ineffective for some versions.

With test suites of "bigrand", suspiciousness results with ENS and ENSω are obtained. Two comprehensive rankings are obtained without dependence information, and ten are obtained with dependence information by using five different strategies. The average comprehensive ranking of fault of each fault version with each method is shown in Figure 3.

With data dependence information of failed executions, ENSD and ENSωD have better performance, and gain an average increase of 5.6% and 7.6% respectively. With five different comprehensive ranking strategies, ENSD, ENSCD, ENSDC, ENSRD and ENSRCD increase the average ranking about 5.6%, 4.3%, 4.3%, 5.1% and 4.2% respectively over ENS. And ENSωD, ENSωCD, ENSωDC, ENSωRD and ENSωRCD increase the average ranking about 7.6%, 6.5%, 6.5%, 7.1% and 6.4% respectively. Our comprehensive statement ranking algorithm performs better than other metrics for version 5 where the fault is missing partial code in assignment statement and version 34 where the fault is in if condition.

By using dependence information (except ENS and ENSω) and suspiciousness results with test suites of "bigcov", the average comprehensive ranking of each version is obtained, as shown in Figure 4.

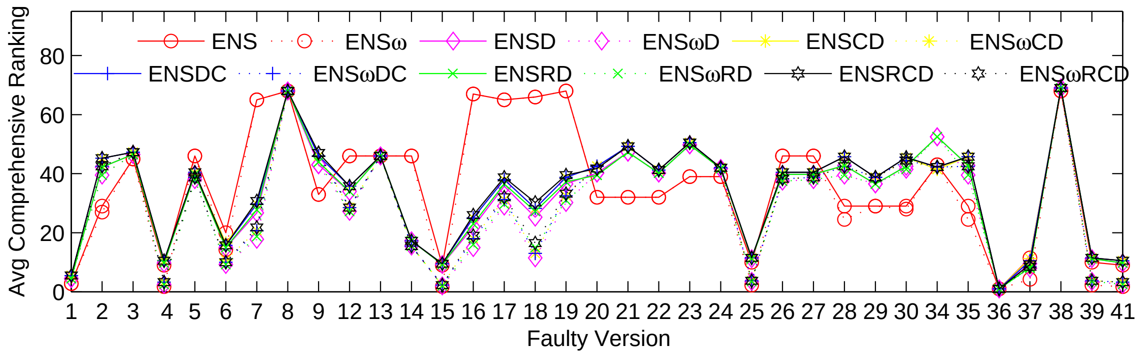

FIGURE 4. The average comprehensive ranking of fault with "bigcov" suites

ENSD, ENSCD, ENSDC, ENSRD and ENSRCD gain an average increase of 4.5%, 3.1%, 3.1%, 4.4% and 2.9% respectively, and ENSD, ENSωCD, ENSωDC, ENSωRD and ENSωRCD gain an average increase of 6.5%, 5.1%, 5.1%, 6.2% and 4.9% respectively. Each method with different performance all has good performance for many versions, especially for versions 7, 16, 17, 18 and 19.

6. Conclusions.

We propose a suspiciousness method ENS based on failed execution and successful non-execution spectra. And a weighted suspiciousness method ENSω is designed by using our proposed execution trace self-information. With the suspiciousness ranking result, a comprehensive statement ranking algorithm is designed to assist locating software fault. Experiments show that fault mostly has higher ranking with test suites of different types. ENS and ENSω outperform other methods, especially ENSω, and fewer statements have to be examined for fault localization. Furthermore, the fault ranking can be increased by using the comprehensive statement ranking algorithm. And fewer statements need to be examined according to the ranking until the fault is located.

To locate fault in large and complex software system effectively, it should be considered in the future work how to design fault localization method to combine fault localization method of statement granularity and method of function granularity.

Acknowledgment. This work was supported by the National Natural Science Foundation of China (No. F020512, F020204), the Natural Science Foundation of Hebei Province (No. F2014203152) and the Education Department of Jilin Province [2016]96.

REFERENCES

[1] J. Ren, C. Wang, H. He and J. Dong, Identifying influential nodes in weighted network based on evidence theory and local structure, *International Journal of Innovative Computing, Information and Control*, vol.11, no.5, pp.1765-1777, 2015.

[2] S. Yoo and M. Harman, Regression testing minimization, selection and prioritization: A survey, *Software Testing, Verification and Reliability*, vol.22, no.2, pp.67-120, 2012.

[3] C. Fang, Z. Chen, K. Wu and Z. Zhao, Similarity-based test case prioritization using ordered sequences of program entities, *Software Quality Journal*, vol.22, no.2, pp.335-361, 2014.

[4] P. Rao, Z. Zheng, T. Y. Chen, N. Wang and K. Cai, Impacts of test suite's class imbalance on spectrum-based fault localization techniques, *Proc. of the 13th International Conference on Quality Software*, Najing, China, pp.260-267, 2013.

[5] W. Wong, Y. Qi, L. Zhao and K. Cai, Effective fault localization using code coverage, *Proc. of the 31st Annual International Computer Software and Applications Conference*, Beijing, China, pp.449-456, 2007.

[6] L. Naish, H. J. Lee and K. Ramamohanarao, A model for spectra-based software diagnosis, *ACM Trans. Software Engineering and Methodology*, vol.20, no.3, pp.1-32, 2011.

[7] T. Janssen, R. Abreu and A. J. C. Van Gemund, Zoltar: A spectra-based fault localization tool, *Proc. of the 2009 ESEC/FSE Workshop on Software Integration and Evolution Runtime*, Amsterdam, The Netherlands, pp.23-29, 2009.

[8] P. Daniel, K. Y. Sim and S. Seol, Improving spectrum-based fault-localization through spectra cloning for fail test cases, *Contemporary Engineering Sciences*, vol.7, no.14, pp.677-682, 2014.

[9] X. Zhang and R. Gupta, Whole execution traces and their applications, *ACM Trans. Architecture and Code Optimization*, vol.2, no.3, pp.301-334, 2005.