# Extending the Theoretical Fault Localization Effectiveness Hierarchy with Empirical Results at Different Code Abstraction Levels*

Chung Man Tang, W.K. Chan, Y.T. Yu†
Department of Computer Science
City University of Hong Kong
Hong Kong
c.m.tang@my.cityu.edu.hk, {wkchan, csytyu}@cityu.edu.hk

***Abstract*—Spectrum-based fault localization techniques are semi-automated program debugging techniques that address the bottleneck of finding suspicious program locations for diagnosis. They assess the fault suspiciousness of individual program locations based on the code coverage data achieved by executing the program under debugging over a test suite. A program location can be viewed at different abstraction levels, such as a statement in the source code or an instruction compiled from the source code. In general, a program location at one code abstraction level can be transformed into zero to more program locations at another abstraction level. Although programmers usually debug at the source code level, the code is actually executed at a lower level. It is unclear whether the same techniques applied at different code abstraction levels may achieve consistent results. In this paper, we study a suite of spectrum-based fault localization techniques at both the source and instruction code levels in the context of an existing theoretical hierarchy to assess whether their effectiveness is consistent across the two levels. Our study extends the theoretical hierarchy with empirically validated relationships across two code abstraction levels toward an integration of the theory and practice of fault localization.**

***Keywords — code abstraction level; fault localization assessment; program debugging; program testing***

## I. INTRODUCTION

Fault localization refers to the identification of fault locations in a program. This task is known to be tedious and time-consuming if it is to be done entirely manually. Many automated or semi-automated techniques, such as program slicing [19][36], delta debugging [40], and *spectrum-based fault localization* (SBFL) [18], have been developed to alleviate the difficulty in fault localization. Recently, there is much research progress in SBFL. A typical SBFL technique, such as *Tarantula* [18], uses a formula involving the execution coverage statistics of the faulty program over a set of test cases (in which at least one test case reveals a failure) to compute the *suspiciousness score* of each program entity, such as statement [1], predicate [43], and subpath [4]. We refer to such a formula as a *SBFL formula*. The entities in the faulty program are then ranked by their suspiciousness scores. The higher the rank of the faulty program entity, the more effective a SBFL technique is. The effectiveness of a SBFL technique thus depends primarily on the formula it uses to compute the suspiciousness of the program entities.

*This work is supported in part by the ECS and GRF of the Research Grants Council of Hong Kong (project numbers 111313, 125113, and 123512).
†Contact author.

Many research studies [17][18][34][39][42] have been conducted to improve or assess the effectiveness of SBFL formulas in fault localization. In particular, Naish et al. [29] compared the effectiveness of a set of SBFL formulas by means of a theoretical model. Xie et al. [38] further examined the theoretical equivalence of these formulas and showed that some of them are equally effective. In this way, Xie et al. classified a set of SBFL formulas into six groups, denoted as ER1–ER6, respectively. They proved that, under certain assumptions, formulas used in some of these groups are never less effective than those in some other groups. Accordingly, they arranged these six groups in a hierarchy as shown in Figure 1. To simplify our presentation, we refer to this hierarchy as the *ER hierarchy*, the SBFL formulas that belong to the ER hierarchy as the *ER formulas*, the groups ER1–ER6 as the *ER formula groups* (or simply *ER groups*), and other SBFL formulas proposed in the literature that are studied in this paper as *non-ER formulas*.

In Figure 1, each node represents an equivalent group of SBFL formulas. A directed edge → from one node $N_1$ to another node $N_2$ (that is, $N_1 \rightarrow N_2$) indicates that for any formula $F_1$ in $N_1$ and any formula $F_2$ in $N_2$, $F_1$ is never less effective than $F_2$. For simplicity, we refer to → as the "*better than*" relation. As such, if two equivalent groups are not related directly or transitively by the directed edges, the formulas in one group are not always *better than* those in the other group, and vice versa. As seen from the figure, both of the nodes ER1 and ER5 do not have any incoming edge. Hence, none of the formulas in these six groups is always *better than* the formulas in either ER1 or ER5.

Our work is inspired by both the ER hierarchy derived from theory and the need to empirically compare SBFL techniques based on their formulas. We also compare the category of ER formulas with the category of non-ER formulas. To the best of our knowledge, there has been no previous systematic study to compare the equivalent groups in the ER hierarchy or the category of ER formulas *versus* that of non-ER formulas between different program entities.

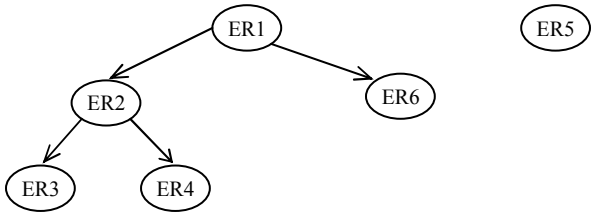



Figure 1. The ER hierarchy of SBFL formulas.

Moreover, we note that even though programmers usually perform debugging at the source code level, the program is actually executed at a lower level of object code instructions. Thus, in this work, we study SBFL formulas using two different program entities: statements at the source code level (or simply the *statement level*) and the instruction codes of the corresponding statements at the object code level (or simply the *instruction level*).

In this study, we extend the theoretical work of Naish et al. [29] and Xie et al. [38] with empirical results of the effectiveness of SBFL formulas. Our study includes a total of 38 SBFL formulas and uses the *Siemens suite* and the *space* program as the experimental subjects. The result shows that the category of non-ER formulas as a whole is statistically more effective than the category of ER formulas, and this is true at both the statement and instruction levels. Within both categories, a SBFL technique is in general more effective when applied at the instruction level than applied at the statement level. Furthermore, we have analyzed the data within each ER equivalent group, which produce empirical findings that enrich and extend the theoretically-derived ER hierarchy. The combined theoretical and empirical results reveal a comprehensive map (see Figure 5) of SBFL techniques which shows how the corresponding SBFL formulas at the two levels in different equivalent groups are related.

The main contribution of this paper is threefold: (1) To the best of our knowledge, this paper is the *first* that extends the theoretical ER hierarchy with empirical results leading to a comprehensive map among techniques at two different code abstraction levels (statement level versus instruction level) of program entities. (2) It demonstrates that the effectiveness of ER formulas in one abstraction level can be consistent with that in another abstraction level, providing empirical support of the applicability of SBFL formulas at different code abstraction levels. (3) It provides empirical evidence which confirms that non-ER formulas can be more effective than ER formulas. It shows that existing theoretical results on the ER hierarchy should be extended to consider other SBFL techniques whose formulas are more effective. Moreover, future SBFL experiments should consider including the comparison with non-ER formulas.

Section II reviews some preliminaries of this work and introduces a motivating example to illustrate the difference between statement-level SBFL and instruction-level SBFL. Section III states the research questions and describes the process and setting of the empirical evaluation. Section IV presents the evaluation results and discusses the findings of the experiment. Section V outlines the related work. Finally, Section VI concludes the paper.

## II. Preliminaries and Motivating Example

### A. Preliminaries

SBFL formulas utilize four variables (which we call *SBFL variables*) to estimate the fault suspiciousness of each program entity (such as a statement) by computing its *suspiciousness score*, denoted by *susp*. The four SBFL variables are as follows.

- passed tests in which it was executed, $a_{ep}$
- failed tests in which it was executed, $a_{ef}$
- passed tests in which it was not executed, $a_{np}$
- failed tests in which it was not executed, $a_{nf}$

For example, one of the 33 SBFL formulas studied in Abreu et al. [1] is known as *Jaccard* [15]:

$$susp_{jaccard} = \frac{a_{ef}}{a_{ef} + a_{nf} + a_{ep}}$$

The suspiciousness scores of all the program entities in the program under debugging can be ordered into *ranks* by using a certain *ranking scheme* to prioritize the manual code inspection sequence. A ranking scheme is one which assigns a rank value to each item (in this case a suspiciousness score) in an ordered list. The higher the suspiciousness score of an entity, the lower is its rank value.

Program source statements are a common type of program entities studied in fault localization research. Suppose $S_1$, $S_2$, $S_3$ and $S_4$ are four source statements of a program and their suspiciousness scores are 0.4, 0.5, 0.8, and 0.5, respectively. Using the *modified competition ranking scheme* (also called the "1334" scheme) [28], for instance, the most suspicious statement $S_3$ is assigned the rank value 1 ("first") and the next two most (equally) suspicious statements $S_2$ and $S_4$ are both assigned the rank value 3 ("joint third"), while the least suspicious statement $S_1$ is assigned the rank value 4 ("forth"). In principle, any ranking scheme, as long as it is consistently applied, can be used. In this empirical study, we use the modified competition ranking scheme.

### B. Motivating Example

Although program source statements are the common entities studied in fault localization research, SBFL formulas can also be applied to compute the fault suspiciousness of object code instructions. Figure 2 illustrates the application of the SBFL formula, *Jaccard*, to a program segment at both the statement level (SL) and the instruction level (IL).

Figure 2(a) shows an excerpt of the faulty program *tcas* version 26, which is part of the *Siemens suite* extracted from the *Software-artifact Infrastructure Repository* (SIR) [8]. This program was written in the C language and consists of 173 lines of source statements. The code excerpt shows 10 consecutive lines of the function `alt_sep_test()`, referred to by the line numbers `112`–`121`, in which line `118` is the faulty statement. In this example, each statement is generally transformed into several object code instructions.

We executed the program against the test pool in SIR and obtained the values of the four variables, $a_{ep}$, $a_{ef}$, $a_{np}$, and $a_{nf}$, at both SL and IL. Figure 2(b) shows the values of these variables and the *Jaccard* suspiciousness scores computed for the source statements (at the SL) in lines `118`–`120` and their corresponding instructions (at the IL). Also shown in Figure 2(b) are the address and category of each individual instruction, together with the ranks of each source statement at the SL and the IL, as to be further explained below.

| Line No. | *tcas* program version 26 |
|---|---|
| 112 | `int alt_sep_test()` |
| 113 | `{` |
| 114 | `bool enabled, tcas_equipped, intent_not_known;` |
| 115 | `bool need_upward_RA, need_downward_RA;` |
| 116 | `int alt_sep;` |
| 117 | |
| **118** | **`enabled = High_Confidence && (Cur_Vertical_Sep > MAXALTDIFF); /* faulty statement */`** |
| **119** | **`tcas_equipped = Other_Capability == TCAS_TA;`** |
| **120** | **`intent_not_known = Two_of_Three_Reports_Valid && Other_RAC == NO_INTENT;`** |
| 121 | |

(a) A code excerpt of the faulty program *tcas* version 26 extracted from SIR [8]

| Line No. | Statement Level (SL) | | | | | | Instruction Level (IL) | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $a_{ep}$ | $a_{ef}$ | $a_{np}$ | $a_{nf}$ | *Jaccard* | Rank* | Address | Category | $a_{ep}$ | $a_{ef}$ | $a_{np}$ | $a_{nf}$ | *Jaccard* | Rank* |
| **118** **Faulty** | 1567 | 11 | 30 | 0 | **0.0070** | **58** | 4196366 | DATAXFER | 1567 | 11 | 30 | 0 | 0.0070 | **31** |
| | | | | | | | 4196372 | LOGICAL | 1567 | 11 | 30 | 0 | 0.0070 | |
| | | | | | | | 4196374 | COND_BR | 1567 | 11 | 30 | 0 | 0.0070 | |
| | | | | | | | 4196376 | DATAXFER | 1194 | 11 | 403 | 0 | 0.0091 | |
| | | | | | | | 4196382 | BINARY | 1194 | 11 | 403 | 0 | 0.0091 | |
| | | | | | | | 4196387 | COND_BR | 1194 | 11 | 403 | 0 | 0.0091 | |
| | | | | | | | 4196389 | DATAXFER | 1087 | 11 | 510 | 0 | **0.0100** | |
| | | | | | | | 4196394 | UNCOND_BR | 1087 | 11 | 510 | 0 | **0.0100** | |
| | | | | | | | 4196396 | DATAXFER | 480 | 0 | 1117 | 11 | 0.0000 | |
| | | | | | | | 4196401 | DATAXFER | 1567 | 11 | 30 | 0 | 0.0070 | |
| **119** | 1567 | 11 | 30 | 0 | **0.0070** | **58** | 4196404 | DATAXFER | 1567 | 11 | 30 | 0 | **0.0070** | **58** |
| | | | | | | | 4196410 | BINARY | 1567 | 11 | 30 | 0 | **0.0070** | |
| | | | | | | | 4196413 | BITBYTE | 1567 | 11 | 30 | 0 | **0.0070** | |
| | | | | | | | 4196416 | DATAXFER | 1567 | 11 | 30 | 0 | **0.0070** | |
| | | | | | | | 4196419 | DATAXFER | 1567 | 11 | 30 | 0 | **0.0070** | |
| **120** | 1567 | 11 | 30 | 0 | **0.0070** | **58** | 4196422 | DATAXFER | 1567 | 11 | 30 | 0 | 0.0070 | **32** |
| | | | | | | | 4196428 | LOGICAL | 1567 | 11 | 30 | 0 | 0.0070 | |
| | | | | | | | 4196430 | COND_BR | 1567 | 11 | 30 | 0 | 0.0070 | |
| | | | | | | | 4196432 | DATAXFER | 695 | 6 | 902 | 5 | 0.0085 | |
| | | | | | | | 4196438 | LOGICAL | 695 | 6 | 902 | 5 | 0.0085 | |
| | | | | | | | 4196440 | COND_BR | 695 | 6 | 902 | 5 | 0.0085 | |
| | | | | | | | 4196442 | DATAXFER | 440 | 4 | 1157 | 7 | **0.0089** | |
| | | | | | | | 4196447 | UNCOND_BR | 440 | 4 | 1157 | 7 | **0.0089** | |
| | | | | | | | 4196449 | DATAXFER | 1127 | 7 | 470 | 4 | 0.0062 | |
| | | | | | | | 4196454 | DATAXFER | 1567 | 11 | 30 | 0 | 0.0070 | |

* *Note: The statements are ranked based on the Jaccard suspiciousness scores of all the 173 statements in the tcas program. The higher the suspiciousness score (shown in the columns labelled '**Jaccard**'), the smaller is the rank value (shown in the columns labelled '**Rank**').*

(b) Some code coverage statistics of statements in lines `118–120` and the corresponding *Jaccard* suspiciousness scores and ranks

Figure 2. An example illustrating the suspiciousness scores and rank values obtained at the two abstraction levels: SL and IL.

At the SL, the values of the SBFL variables for all three statements in lines `118–120` are found to be the same, thus yielding the same suspiciousness score (**0.0070**), as shown in Figure 2(b). Hence they also have the same rank value (**58**), based on ordering the *Jaccard* suspiciousness scores of all the 173 statements in the program using the modified competition ranking scheme.

At the IL, the three statements in lines `118–120` were translated into 25 instructions with addresses from `4196366` to `4196454`, each instruction having its own set of coverage information. We computed the suspiciousness score of each instruction by applying the same *Jaccard* formula at the IL instead of at the SL. We then used the *highest* of the suspiciousness scores (shown in boldface) of all the instructions corresponding to each statement to estimate the statement's suspiciousness. For example, corresponding to the source statement in line `118`, the suspiciousness scores of all the instructions (with addresses `4196366–4196401`) range from 0.0000 to 0.0100. The score **0.0100** is the highest at the instructions of addresses `4196389` and `4196394`. Hence, we used **0.0100** as the estimate of suspiciousness of the source statement in line `118`. In the same way, the suspiciousness estimates of the statements in lines `119` and `120` were computed as **0.0070** and **0.0089**, respectively. By ordering these suspiciousness estimates of all the 173 source statements of the program, the statements in lines `118–120` were ranked as **31**, **58**, and **32**, respectively, as shown in boldface in the rightmost column in Figure 2(b).

In this example, the *Jaccard* formula is more effective at the IL as the rank value (**31**) of the faulty statement (in line `118`) at the IL is much smaller than that (**58**) at the SL. It shows that the same SBFL formula computed at different abstraction levels may result in significantly different fault localization effectiveness.

## III. EMPIRICAL EVALUATION

### A. Research Questions

Our empirical study examines three research questions:

RQ1. At each of the two abstraction levels, which category of SBFL formulas is more effective, *ER formulas* or *non-ER formulas*?

RQ2. When the same SBFL formula is used, at which of the two abstraction levels is the formula more effective? Is this result consistent among different formulas?

RQ3. Consider ER formulas only. Suppose that formula F1 is known theoretically to be *better than* formula F2. Suppose that both F1 and F2 at one particular abstraction level (A1) are more effective than the same formulas at another abstraction level (A2). Is F1 at level A2 more effective than F2 at level A1?

### B. Subject Programs

We adopted 7 programs in the *Siemens suite* and the *space* program as our subject programs. Each program in the *Siemens suite* consists of 7 to 41 faulty versions and each version contains a single seeded fault. There are a total of 132 faulty versions in the *Siemens suite*. Since these *Siemens suite* programs are relatively small (each version consisting of 172 to 566 lines of code), we supplement our set of subject programs with a larger program called *space* (also extracted from SIR [8]) which consists of more than 9000 lines. There are 38 faulty versions of the *space* program and each version contains a single real-life fault.

As SBFL formulas generally require non-zero failed test runs, we excluded 5 faulty versions in the *Siemen suite* and 3 faulty versions of the *space* program from the experiment because none of the accompanied test cases reveal the faults in these versions. All in all, 127 faulty versions in the *Siemens suite* and 35 faulty versions of the *space* program were adopted in the experiment. A summary of all the 8 subject programs is shown in Table I. To contrast between SL and IL, we executed all test cases in the test pool instead of executing subsets of the test pool as test suites.

### C. SBFL Formulas

Naish et al. [29] compared a set of 33 SBFL formulas and Xie et al. [38] further analyzed over 30 of them and some others. Our experiment used all the 33 formulas from Naish et al. and 5 additional formulas from Xie et al. The complete set studied in this paper includes 22 ER formulas and 16 non-ER formulas, as listed in Table II and Table III, respectively.

### D. Execution Context

We used the *Pin* tool [25] to instrument the subject programs and extract their execution contexts at runtime as the test cases were executed. The context of each test execution extracted for this research includes the line numbers of the source statements, the instruction addresses, the number of times that the instruction is executed. Together with the information of whether the tests revealed the fault in the faulty version being executed, all the code coverage data required could then be collected for computing the suspiciousness scores at both SL and IL.

TABLE I. SUMMARY INFORMATION OF THE SUBJECT PROGRAMS

| Program | | Size | No. of Faulty Versions | No. of Fault-revealing Versions | No. of Test Cases |
|---|---|---|---|---|---|
| *Siemens Suite* | printtokens | 563–564 | 7 | 7 | 4130 |
| | printtokens2 | 504–510 | 10 | 10 | 4115 |
| | replace | 562–566 | 32 | 29 | 5542 |
| | schedule | 411–414 | 9 | 9 | 2650 |
| | schedule2 | 307–308 | 10 | 9 | 2710 |
| | tcas | 172–179 | 41 | 40 | 1608 |
| | totinfo | 406 | 23 | 23 | 1052 |
| *space* | | 9118–9126 | 38 | 35 | 13585 |
| | | **Total** | 170 | 162 | |

### E. Performance Metrics

*1) Suspiciousness score* (*susp*): Using a SBFL formula, *susp* is computed to estimate *the degree of suspiciousness of a program entity being faulty*, as explained in Section II.A.

*2) Code Examination Effort* (EXAM): To evaluate the effectiveness of a SBFL formula in localizing the fault in a faulty program, we adopted the metric called *code examination effort*, a.k.a. *expense* or *EXAM score* [16], which reflects the effort spent by programmers in locating the faults. The EXAM score is calculated as follows.

$$\text{EXAM score} = \frac{\text{Rank of the faulty statement}}{\text{Total number of statements}}$$

The smaller the EXAM score, the more effective (accurate) the SBFL formula is *with respect to the faulty program* under debugging. For example, for the program in Figure 2 which consists of 173 statements, the rank of the faulty statement at the SL is 58 and that at the IL is 31. Hence the EXAM scores are 58/173 = 0.34 at the SL and 31/173 = 0.18 at the IL. In this example, the formula is more effective at the IL.

*3) Area Under Curve* (AUC): To assess the aggregate effectiveness of a SBFL formula in localizing the faults in a set of faulty versions of a program, we analyzed the data by plotting the cumulative percentage of faulty versions revealed versus the EXAM score as the latter progresses from 0 to 100. We then computed the *Area Under Curve* (AUC) of the plot as an aggregate measure of the effectiveness of the formula *with respect to all the faulty versions*. A larger value of the AUC metric for a formula means that more cumulative faults are found by examining the same amount of code. Hence, the larger the value of AUC, the more effective is the formula with respect to the set of faulty versions under study.

### F. Experiment Setting

We executed all the faulty versions against all the test cases in the associated test pool to obtain the execution contexts. Then we computed the suspiciousness scores, EXAM scores, and the AUC values of each SBFL formula at both SL and IL. In this section, we describe the details of the environment, tools and procedure of the experiment.

TABLE II. SBFL ER FORMULAS USED IN THIS STUDY

| Name | Group | ER Formulas | Ref. |
|---|---|---|---|
| Naish1 | ER1 | $\begin{cases} -1, \text{ if } a_{ef} < F \\ P - a_{ep}, \text{ if } a_{ep} = F \end{cases}$ | [29] |
| Naish2 | | $a_{ef} - \frac{a_{ep}}{a_{ep} + a_{np} + 1}$ | [29] |
| Jaccard | ER2 | $\frac{a_{ef}}{a_{ef} + a_{nf} + a_{ep}}$ | [15] |
| Anderberg | | $\frac{a_{ef}}{a_{ef} + 2(a_{nf} + a_{ep})}$ | [2] |
| Sørensen-Dice | | $\frac{2a_{ef}}{2a_{ef} + a_{nf} + a_{ep}}$ | [7] |
| Dice | | $\frac{2a_{ef}}{a_{ef} + a_{nf} + a_{ep}}$ | [7] |
| Goodman | | $\frac{2a_{ef} - a_{nf} - a_{ep}}{2a_{ef} + a_{nf} + a_{ep}}$ | [13] |
| Tarantula | ER3 | $\frac{\frac{a_{ef}}{a_{ef}+a_{nf}}}{\frac{a_{ef}}{a_{ef}+a_{nf}} + \frac{a_{ep}}{a_{ep}+a_{np}}}$ | [18] |
| $q_e$ | | $\frac{a_{ef}}{a_{ef} + a_{ep}}$ | [21] |
| CBI Inc. | | $\frac{a_{ef}}{a_{ef} + a_{ep}} - \frac{a_{ef} + a_{nf}}{a_{ef} + a_{nf} + a_{ep} + a_{np}}$ | [23] |
| Wong2 | ER4 | $a_{ef} - a_{ep}$ | [37] |
| Hamann | | $\frac{a_{ef} + a_{np} - a_{nf} - a_{ep}}{a_{ef} + a_{nf} + a_{ep} + a_{np}}$ | [24] |
| Simple Matching | | $\frac{a_{ef} + a_{np}}{a_{ef} + a_{nf} + a_{ep} + a_{np}}$ | [27] |
| Sokal | | $\frac{2(a_{ef} + a_{np})}{2(a_{ef} + a_{np}) + a_{nf} + a_{ep}}$ | [24] |
| Rogers & Tanimoto | | $\frac{a_{ef} + a_{np}}{a_{ef} + a_{np} + 2(a_{nf} + a_{ep})}$ | [31] |
| Hamming etc. | | $a_{ef} + a_{np}$ | [14] |
| Euclid | | $\sqrt{a_{ef} + a_{np}}$ | [20] |
| Wong1 | ER5 | $a_{ef}$ | [37] |
| Russell & Rao | | $\frac{a_{ef}}{a_{ef} + a_{nf} + a_{ep} + a_{np}}$ | [33] |
| Binary | | $\begin{cases} 0, \text{ if } a_{ef} < F \\ 1, \text{ if } a_{ep} = F \end{cases}$ | [29] |
| Scott | ER6 | $\frac{4a_{ef}a_{np} - 4a_{nf}a_{ep} - (a_{nf} - a_{ep})^2}{(2a_{ef} + a_{nf} + a_{ep})(2a_{np} + a_{nf} + a_{ep})}$ | [35] |
| Rogot1 | | $\frac{1}{2}\left(\frac{a_{ef}}{2a_{ef} + a_{nf} + a_{ep}} + \frac{a_{np}}{2a_{np} + a_{nf} + a_{ep}}\right)$ | [32] |

TABLE III. SBFL NON-ER FORMULAS USED IN THIS STUDY

| Name | Non-ER Formulas | Ref. |
|---|---|---|
| Ample | $\left\lvert \frac{a_{ef}}{a_{ef} + a_{nf}} - \frac{a_{ep}}{a_{ep} + a_{np}} \right\rvert$ | [6] |
| Arithmetic mean | $\frac{2a_{ef}a_{np} - 2a_{nf}a_{ep}}{(a_{ef} + a_{ep})(a_{np} + a_{nf}) + (a_{ef} + a_{nf})(a_{ep} + a_{np})}$ | [32] |
| Cohen | $\frac{2a_{ef}a_{np} - 2a_{nf}a_{ep}}{(a_{ef} + a_{ep})(a_{np} + a_{ep}) + (a_{ef} + a_{nf})(a_{nf} + a_{np})}$ | [5] |
| Fleiss | $\frac{4a_{ef}a_{np} - 4a_{nf}a_{ep} - (a_{nf} - a_{ep})^2}{(2a_{ef} + a_{nf} + a_{ep}) + (2a_{np} + a_{nf} + a_{ep})}$ | [11] |
| Geometric mean | $\frac{a_{ef}a_{np} - a_{nf}a_{ep}}{\sqrt{(a_{ef} + a_{ep})(a_{np} + a_{nf})(a_{ef} + a_{nf})(a_{ep} + a_{np})}}$ | [26] |
| Harmonic Mean | $\frac{(a_{ef}a_{np} - a_{nf}a_{ep})\left((a_{ef} + a_{ep})(a_{np} + a_{nf}) + (a_{ef} + a_{nf})(a_{ep} + a_{np})\right)}{(a_{ef} + a_{ep})(a_{np} + a_{nf})(a_{ef} + a_{nf})(a_{ep} + a_{np})}$ | [32] |
| Kulczynski1 | $\frac{a_{ef}}{a_{nf} + a_{ep}}$ | [24] |
| Kulczynski2 | $\frac{1}{2}\left(\frac{a_{ef}}{a_{ef} + a_{nf}} + \frac{a_{ef}}{a_{ef} + a_{ep}}\right)$ | [24] |
| M1 | $\frac{a_{ef} + a_{np}}{a_{nf} + a_{ep}}$ | [10] |
| M2 | $\frac{a_{ef}}{a_{ef} + a_{np} + 2(a_{nf} + a_{ep})}$ | [10] |
| Ochiai | $\frac{a_{ef}}{\sqrt{(a_{ef} + a_{nf})(a_{ef} + a_{ep})}}$ | [30] |
| Ochiai2 | $\frac{a_{ef}a_{np}}{\sqrt{(a_{ef} + a_{ep})(a_{np} + a_{nf})(a_{ef} + a_{nf})(a_{ep} + a_{np})}}$ | [30] |
| Overlap | $\frac{a_{ef}}{\min(a_{ef}, a_{nf}, a_{ep})}$ | [9] |
| Rogot2 | $\frac{1}{4}\left(\frac{a_{ef}}{a_{ef} + a_{ep}} + \frac{a_{ef}}{a_{ef} + a_{nf}} + \frac{a_{np}}{a_{np} + a_{ep}} + \frac{a_{np}}{a_{np} + a_{nf}}\right)$ | [32] |
| Wong3 | $a_{ef} - h$, where $h = \begin{cases} a_{ep}, \text{ if } a_{ep} \le 2 \\ 2 + 0.1(a_{ep} - 2), \text{ if } 2 < a_{ep} \le 10 \\ 2.8 + 0.001(a_{ep} - 10), \text{ if } a_{ep} > 10 \end{cases}$ | [37] |
| Zoltar | $\frac{a_{ef}}{a_{ef} + a_{nf} + a_{ep} + \frac{10000a_{nf}a_{ep}}{a_{ef}}}$ | [12] |

*1) Environment and tools:* The experiment was conducted in a virtual machine configured with Intel Xeon CPU X5560 @ 2.8GHz×2, 4GB of physical memory and 100GB disk storage. We used Ubuntu 12.04 LTS (64-bit) as the operating system, gcc version 4.6.3 as the compiler, and Pin version 4.13 to instrument the subject programs and extract their execution contexts as the test cases were executed. All formulas and algorithms were implemented in Java and executed within the OpenJDK Runtime Environment (IcedTea6 1.12.5). To analyze the data, we used MATLAB R2103a (8.1.0.604) and Microsoft Excel 2010.

*2) Procedure:* First, we executed all the faulty versions of the subject programs with the entire test pool and recorded the execution contexts of all the test runs.

Next, from the execution context records, the values of the variables in the SBFL formulas were computed at each of the two levels, SL and IL. For instance, for a faulty version of the *printtokens* program with 563 statements, there were 563 entries of statement coverage variables at the SL. At the IL, for a faulty version with 1405 instructions, there were 1405 entries of instruction coverage variables.

Third, at the SL, the suspiciousness scores of the 563 statements were directly computed from the SBFL formulas. At the IL, as explained in the example of Figure 2, the highest suspiciousness score of all the instructions that correspond to each source statement was used as the statement's suspiciousness score. In either abstraction level, the corresponding list of suspiciousness scores was then ranked using the modified competition scheme. Finally, the EXAM scores of each SBFL formula for each faulty program version were computed, followed by the AUC metric.

## IV. Results and Data Analysis

In this section, we report the results of the experiment and then analyze them to examine each research question.

### A. Comparing the Effectiveness between Two Categories: ER Formulas versus non-ER Formulas

Figure 3 consists of two parts. The upper part shows the graphs for the *Siemen suite* and the lower one shows the graphs for the *space* program. Each part contains two graphs. In each graph, the *y*-axis shows the percentage of faulty versions whose faults are located as the percentage of code shown in the *x*-axis is examined. For the outer graph in each part, the range of percentage of code examined is shown from 0 to 100, while the inner graph shows the magnification of the outer one up to 5 percent of code examined. In each graph, four curves were plotted that correspond to the effectiveness of ER formulas at the SL (a black dashed curve), ER formulas at the IL (a blue solid curve), non-ER formulas at the SL (a green dotted curve), and non-ER formulas at the IL (a red solid curve), respectively.

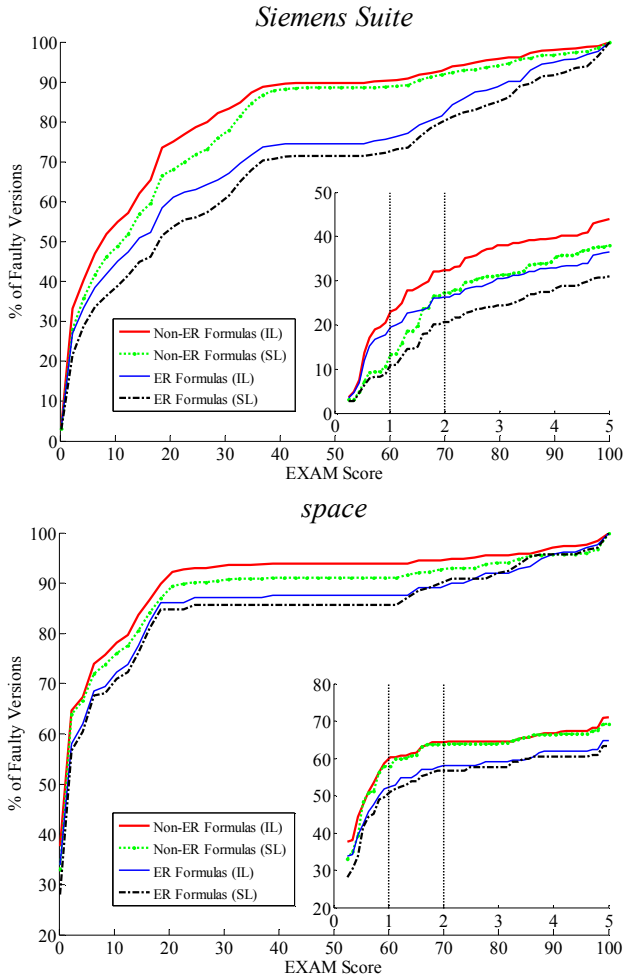



Figure 3. Comparing ER *versus* non-ER formulas and SL *versus* IL.

TABLE IV. AUC VALUES FOR ER AND NON-ER FORMULAS AT SL AND IL

| | *Siemens Suite* | | | The *space* Program | | |
|---|---|---|---|---|---|---|
| | **ER** | **Non-ER** | **% Diff.** | **ER** | **Non-ER** | **% Diff.** |
| **SL** | 0.6819 | 0.8018 | 17.6 | 0.8487 | 0.8855 | 4.3 |
| **IL** | 0.7211 | 0.8293 | 15.0 | 0.8583 | 0.9078 | 5.8 |
| **% Diff.** | 5.7 | 3.4 | | 1.1% | 2.5% | |

We observe from Figure 3 that the curves for non-ER formulas are generally higher than the corresponding curves for ER formulas. Indeed, the AUC values for the former are all correspondingly larger than those for the latter, as shown in Table IV. The differences are 17.6% and 15.0% for the *Siemens suite* at the SL and IL, respectively, and 4.3% and 5.8% for the *space* program at the SL and IL, respectively. Hence, our results show that the category of non-ER formulas is generally better than that of ER formulas.

### B. Comparing the Effectiveness between Two Abstraction Levels: Statement versus Instruction

We also observe from Figure 3 that for both the *Siemens suite* and the *space* program as well as for both ER and non-ER formulas, the (solid) curves for IL are generally slightly higher than the corresponding curves for SL.

Quantitatively, Table IV shows that the AUC values at the IL are higher than those at the SL by 5.7% and 3.4% when applying the ER and non-ER formulas, respectively, to the *Siemens suite*, and by 1.1% and 2.5%, respectively, when applying them to the *space* program. Hence, our results show that in both of the categories of ER formulas and non-ER formulas, the effectiveness is higher when applied at the IL than at the SL.

Figure 4 visually contrasts the effectiveness of individual formulas or ER groups of formulas at the two abstraction levels. The upper part shows a graph for the *Siemens suite* and the lower one shows a graph for the *space* program. Each graph shows pairs of boxplots of EXAM scores, each pair for either a group (ER1–ER6) of formulas or an individual one (e.g., Kulczynski2). Note that formulas within the same ER group are (theoretically) equally effective [29].

We have two observations from Figure 4. First, between the two boxplots of each SBFL formula or formula group, the median EXAM score (shown as a horizontal line inside each box) of the left boxplot (at the SL) is almost invariably higher than that of the right one (at the IL), and this observation holds for both the *Siemens suite* and the *space* program. Because higher EXAM scores indicate lower effectiveness, the same formula (or group) is generally more effective at the IL than at the SL. The differences, however, are smaller for the *space* program than for the *Siemens suite*.

Second, in both graphs, non-ER formulas are generally nearer to the left than ER groups, except for ER1 in the graph for the *Siemens suite*. The formulas (or groups) in Figure 4 are sorted in ascending order of median EXAM scores from left to right. Those nearer the left are generally more effective than those nearer the right. Hence, this observation re-affirms that generally the category of non-ER formulas is more effective than that of ER formulas.

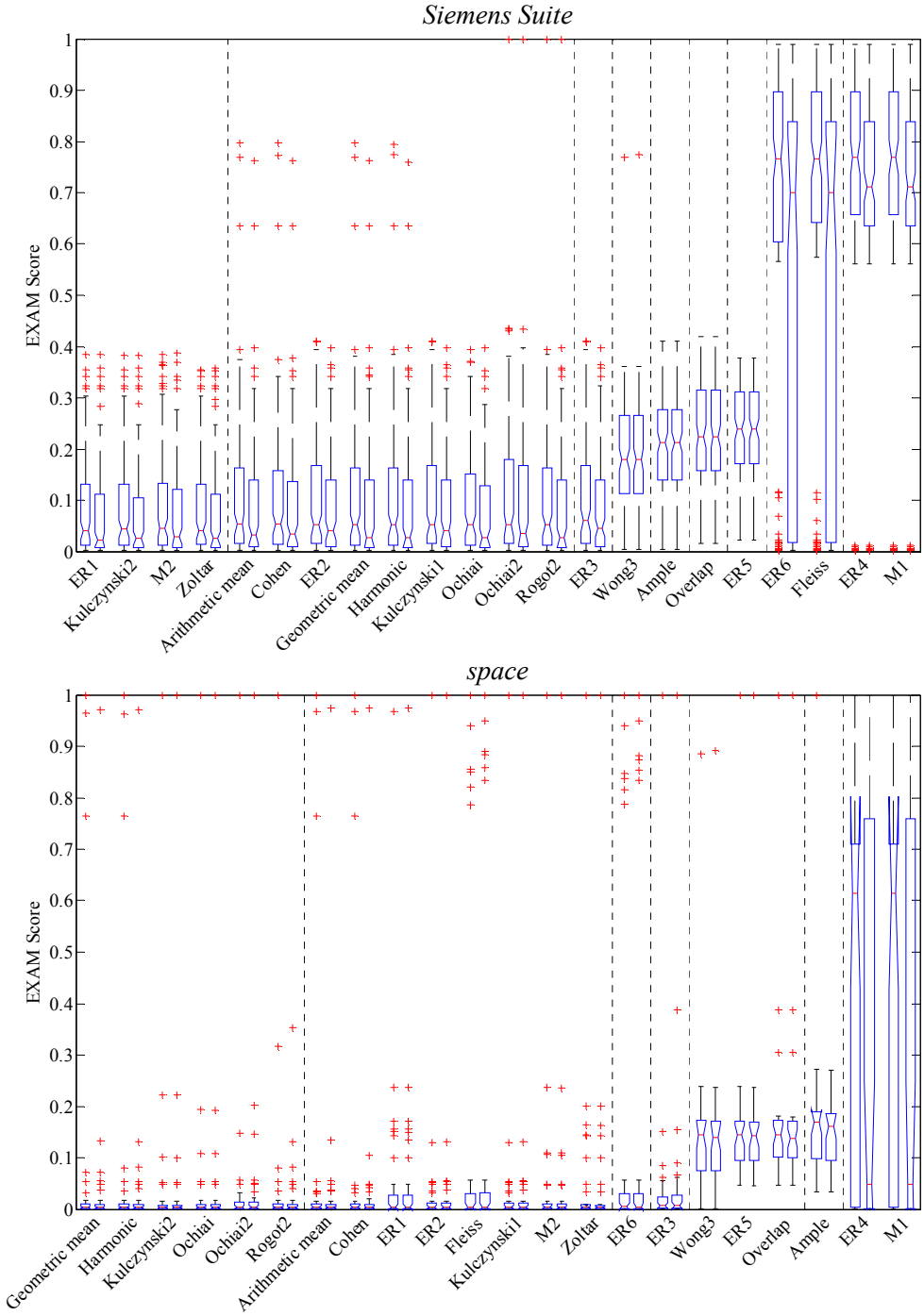
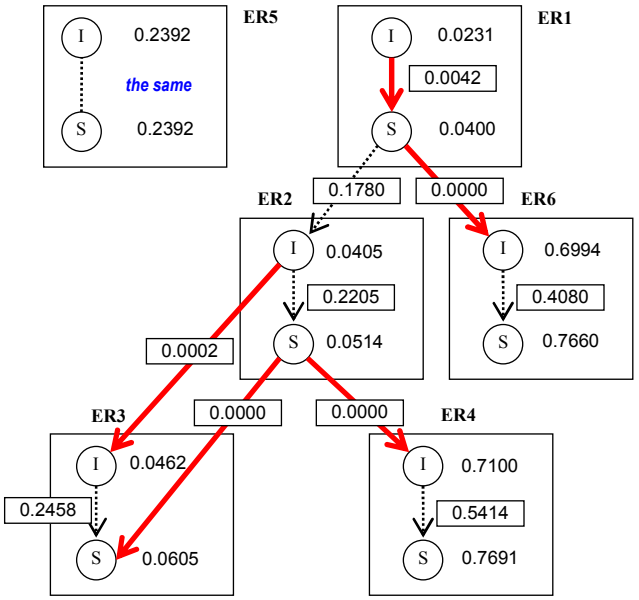


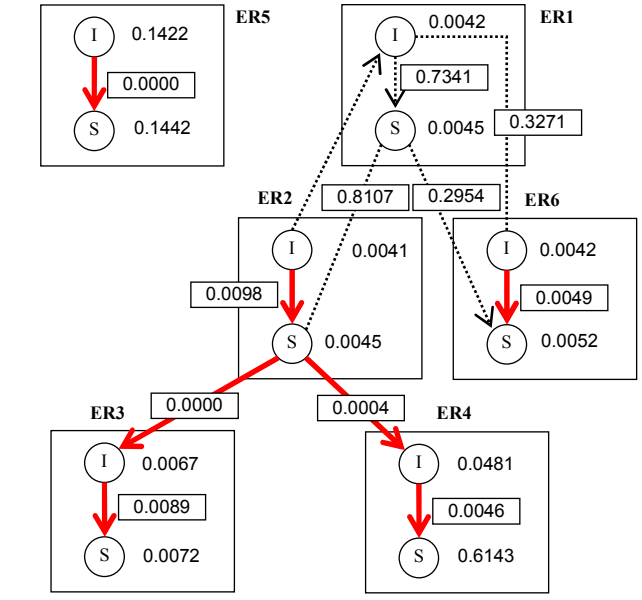

Figure 4. Boxplots of EXAM scores of either individual SBFL formulas or ER groups of formula arranged in ascending order of medians from left to right: The *x*-axis shows the name of the SBFL formulas or formula groups. For each formula or formula group, a pair of boxplots is shown in which the EXAM score distribution at the SL is on the left while that at the IL is on the right. For each boxplot, the horizontal line inside the box shows the median, the boundaries of the box enclose the interquartile range, and the crosses "+" outside the box represent outliers. The formulas or formula groups with the same median EXAM score (within 0.1% tolerance) are grouped together and separated by dashed vertical lines.

### C. *Comparing the Effectiveness within the ER Hierarchy: between Abstraction Levels and Formula Groups*

Figure 5 shows two diagrams, each illustrating an ER hierarchy enhanced with fault localization effectiveness comparison results between the two abstractions levels for (a) the *Siemens suite*, and (b) the *space* program, respectively.




(a) Comparison using the *Siemens suite*




(b) Comparison using the *space* program

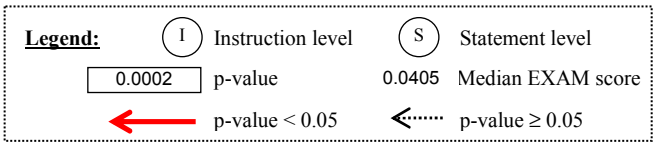



Figure 5. Comparing the effectiveness within the ER hierarchy.

In each diagram of Figure 5, a *square* represents a formula group of ER1–ER6. Within each group, there are two circular nodes labeled with "S" and "I", representing SL and IL, respectively. Each node is associated with the median EXAM score of that particular abstraction level. Some pairs of nodes are connected with edges. An undirected edge (shown as a black dashed line) between two nodes indicates that the nodes are empirically equally effective (with the same median EXAM score). A directed edge from node X to node Y indicates that X is empirically more effective (with a lower EXAM score) than Y, and the edge is annotated with the p-value (enclosed in a rectangle) of a statistical significance test (the Wilcoxon signed rank test) of the difference in effectiveness. When the difference is statistically significant at a 5% level, the edge is drawn as a red solid arrow, otherwise the edge is drawn as a black dashed arrow.

To reduce the complexity of the diagrams, we show only the edges that connect two ER groups so that all nodes can be "traversed" via the edges. For instance, for the *space* program in Figure 5(b), ER2 and ER3 are connected by an edge between the SL node of ER2 and IL node of ER3, which is a minimal edge needed to traverse all nodes in ER2 and ER3. For the *Siemens suite* in Figure 5(a), two edges are drawn to connect the groups ER2 and ER3 so that all nodes in ER3 can be traversed from nodes above ER3.

The Wilcoxon signed rank test is a non-parametric paired test for evaluating the difference between two lists of values. We define a hypothesis for the EXAM scores between a pair of nodes as follows:

$H_0$: There is no significant difference between the nodes

$H_1$: There is significant difference between the nodes

We adopt 0.05 as the level of significance, that is, if a p-value is smaller than 0.05, the null hypothesis $H_0$ is rejected and the alternative hypothesis $H_1$ holds.

We have the following observations from Figure 5.

First, within the same ER group, every IL node is better than the corresponding SL node, with the former having a lower or the same median EXAM score. For the *space* program, all such differences are statistically significant except in ER1. For the *Siemens suite*, the difference in ER1 is statistically significant while there is no difference in ER5.

Second, all directed edges between pairs of ER groups in the hierarchy point in the same direction, with the only exception edge between the IL nodes in ER1 and ER2. This exception edge, however, is *not* statistically significant.

Third, consider all the pairs of ER groups such that one group is "directly above" the other in the ER hierarchy [38].

(a) For the *Siemens suite*, the SL node of the "upper" group in every such pair is better than the IL node of the "lower" group (all statistically significant except for the edge between ER1 and ER2), with the only exception between ER2 and ER3. The IL and SL nodes of ER2 are respectively better than those of ER3, but the SL node of ER2 is not better than the IL node of ER3.

(b) For the *space* program, the SL node of ER2 is better than the IL nodes of ER3 and ER4 (both being statistically significant), but the SL node of ER1 is not better than the IL nodes of ER2 and ER6.

(c) For both the *Siemens suite* and the *space* program, the SL node of ER2 is statistically better than the IL node of ER4. For other such pairs of ER groups, we do not find consistent comparison results between the SL node of the upper group and the IL node of the lower group.

In summary, our observations (1) further affirm that SBFL formulas are generally better at the IL than at the SL, (2) are generally consistent with the theoretical hierarchy in [29] with only one (statistically insignificant) exception, and (3) show that the SL node of an ER group which is directly above another ER group in the ER hierarchy is often, but not always, better than the IL node of the latter group.

### D. Re-examining the Research Questions

We now answer the three research questions (posed in Section III.A) in light of our empirical data and results discussed in Sections IV.A–C above.

*1) RQ1:* The category of non-ER formulas is in general empirically more effective than that of ER formulas when both are applied at the same code abstraction level.

*2) RQ2:* The SBFL formulas are almost invariably more effective when applied at the instruction level than at the statement level. The differences in effectiveness between the two abstraction levels, however, are not as large as those between the categories of ER and non-ER formulas (e.g., see Table IV).

*3) RQ3:* As implied in our answer to RQ2 above, applying the same formula within an ER group at the instruction level is almost invariably more effective than applying at the statement level. When an ER formula in a group is applied at the statement level, the effectiveness is sometimes but not always higher than that of applying another formula at the instruction level in an ER group directly below the former group in the ER hierarchy.

### E. Threats to Validity

The experiment result shows that the effectiveness of a formula is sometimes dependent on the subject programs. A formula may work very well for a program but badly for another program. To minimize the effect of the behavior of individual programs, instead of just comparing the subject programs within the *Siemens suite*, we have included a medium-scale real-life program, *space*, and also considered the overall effectiveness by treating the *Siemens suite* as a whole. We used subject programs written in the C language only. For other programming languages, such as Java, the way of instrumentation and the execution context information extracted may be quite different and, therefore, our finding may not be generalizable to them. The actual instructions and their coverage extracted from a statement depend on the compiler, its parameters, system platform and execution environment, etc. When these factors vary, it is unknown how effective the SBFL formulas will be at different code abstraction levels. Nevertheless, code coverage information that can be obtained at the instruction level is undoubtedly finer than that at the statement level. Whether the finer coverage information may translate into higher effectiveness of SBFL formulas will need to be further experimented with other combinations of environmental parameters and factors.

## V. Related Work

Li et al. [22] empirically compared the effectiveness of bytecode instrumentation and source code instrumentation with various code coverage tools. Our investigation of effectiveness of SBFL techniques at the instruction level is partly inspired by their work. Santelices et al. [34] evaluated three types of program coverage entities, namely, statements, branches and data dependencies, and concluded that different entities worked better in localizing different faults.

Numerous SBFL formulas have been proposed as listed in Table II and Table IV of this paper. For brevity, we do not explicitly review them. Methods [3][4][17][37][39][42] to improve effectiveness of statistical debugging have been reported. Naish et al. [29] and Xie et al. [38] performed comprehensive studies on many SBFL formulas and their results have been introduced in Section I of this paper. Their theoretical work has laid a solid foundation that motivated this experiment.

Many fault localization techniques other than SBFL have also been proposed in the literature. For instance, program slicing [19][36] attempts to extract relevant program statements to include only the fault inducing ones. Delta debugging [40] removes irrelevant elements of a failure inducing input aiming to obtain the minimal input for reproducing the fault. Predicate switching [41] changes the predicate outcomes of conditional branches to find out whether the program will still produce failures.

## VI. Conclusion

In this paper, we have empirically examined 22 formulas in the ER hierarchy and 16 non-ER formulas at both the statement level and the instruction level. We have found that the category of non-ER formulas is more effective than that of the formulas in the ER hierarchy. Thus, research on theoretically comparing SBFL formulas should be further conducted because there are empirical evidences that non-ER formulas can outperform those in the existing theoretical hierarchy. We have also enhanced the ER hierarchy with empirical results to reveal novel and fine-grained relationships among ER groups. Our experiment has shown that the effectiveness of ER formulas in one code abstraction level can largely be consistently projected into another abstraction level.

To the best of our knowledge, our paper has presented the first work on combining the theoretical ER hierarchy and empirical data at multiple code abstraction levels. Moreover, we have shown that applying the same formula at the instruction level can often be more effective than applying it at the statement level. Further experiments with different settings should be performed to investigate to what extent the same results can be generalized.

## REFERENCES


[1] R. Abreu, P. Zoeteweij, R. Golsteijn, and A.J.C. van Gemund. A practical evaluation of spectrum-based fault localization. *Journal of Systems and Software*, 82(11):1780–1792, 2009.

[2] M.R. Anderberg. *Cluster analysis for applications*. Academic Press, 1973.

[3] P. Arumuga Nainar, T. Chen, J. Rosin, and B. Liblit. Statistical debugging using compound Boolean predicates. In *Proceedings of International Symposium on Software Testing and Analysis*, pp. 5–15, 2007.

[4] T.M. Chilimbi, B. Liblit, K. Mehra, A.V. Nori, and K. Vaswani. HOLMES: Effective statistical debugging via efficient path profiling. In *Proceedings of the International Conference on Software Engineering*, pp. 34–44, 2009.

[5] J. Cohen. A coefficient of agreement for nominal scales. *Educational and Psychological Measurement*, 20:37–46, 1960.

[6] V. Dallmeier, C. Lindig, and A. Zeller. Lightweight defect localization for Java. In *Proceedings of the European Conference on Object-Oriented Programming*, pp. 528–550, 2005.

[7] L. Dice, Measures of the amount of ecologic association between species. *Ecology*, 26(3):297–302, 1945.

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

[9] M. Dunham. *Data Mining: Introductory and Advanced Topics*. Prentice-Hall, 2002.

[10] B. Everitt. *Graphical Techniques for Multivariate Data*. North-Holland. 1978.

[11] J. Fleiss. Estimating the accuracy of dichotomous judgments. *Psychometrika*, 30(4):469–479, 1965.

[12] A. Gonzalez. *Automatic Error Detection Techniques Based on Dynamic Invariants*. M.S. dissertation. Delft University of Technology, The Netherlands, 2007.

[13] L. Goodman and W. Kruskal. Measures of association for cross-classifications. *Journal of the American Statistical Association*, 49 (Part I):732–764, 1954.

[14] R. Hamming. Error detecting and error correcting codes. *Bell System Technical Journal*, 29(2):147–160, 1950.

[15] P. Jaccard. *Etude Comparative de la Distribution Florale dans une Portion des Alpes et du Jura*. Impr. Corbaz, 1901.

[16] D. Jeffrey, N. Gupta, and R. Gupta. Fault localization using value replacement. In *Proceedings of the ACM SIGSOFT International Symposium on Software Testing and Analysis*, pp. 167–178, 2008.

[17] B. Jiang, Z. Zhang, W.K. Chan, T.H. Tse, and T.Y. Chen. How well does test case prioritization integrate with statistical fault localization? *Information and Software Technology*, 54(7):739–758, 2012.

[18] J.A. Jones and M.J. Harrold. Empirical evaluation of the Tarantula automatic fault-localization technique. In *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering*, pp. 273–282, 2005.

[19] B. Korel and J. Laski. Dynamic slicing of computer programs. *Journal of Systems and Software*, 13(3):187–195, 1990.

[20] E. Krause. Taxicab Geometry. *Mathematics Teacher*, 66(8):695–706, 1973.

[21] H. Lee, L. Naish, and K. Ramamohanarao. Study of the relationship of bug consistency with respect to performance of spectra metrics. In *Proceedings of the International Conference on Computer Science and Information Technology*, pp. 501–508, 2009.

[22] N. Li, X. Meng, J. Offutt, and L. Deng. Is bytecode instrumentation as good as source code instrumentation? An empirical study with industrial tools (Experience Report). In *Proceedings of International Symposium on Software Reliability Engineering*, pp. 380–389, 2013

[23] B. Liblit, M. Naik, A.X. Zheng, A. Aiken, and M.I. Jordan. Scalable statistical bug isolation. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation*, pp. 15–26, 2005.

[24] F. Lourenco, V. Lobo, and F. Bação. Binary-based similarity measures for categorical data and their application in Self-Organizing Maps. In *Proceedings of the Conference on Classification and Analysis of Data* (*JOCLAD 2004*), pp. 121–138, 2004.

[25] C.K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V.J. Reddi, and K. Hazelwood. Pin: Building customized program analysis tools with dynamic instrumentation. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation*, pp. 190–200, 2005.

[26] A. Maxwell and A. Pilliner. Deriving coefficients of reliability and agreement for ratings. *British Journal of Mathematical and Statistical Psychology*, 21(1):105–16, 1968.

[27] A. Meyer, A. Garcia, A. Souza, and C. Souza Jr. Comparison of similarity coefficients used for cluster analysis with dominant markers in maize (Zea mays L). *Genetics and Molecular Biology*, 27(1):83–91, 2004.

[28] S.K. Mishra. The Most Representative Composite Rank Ordering of Multi-Attribute Objects by the Particle Swarm Optimization. 2009, Available at http://ssrn.com/abstract=1326386

[29] L. Naish, H.J. Lee, and K. Ramamohanarao. A model for spectra-based software diagnosis. *ACM Transactions on Software Engineering and Methodology*, 20(3), Article no. 11, 2011.

[30] A. Ochiai. Zoogeographic studies on the soleoid fishes found in Japan and its neighbouring regions. *Bulletin of the Japanese Society for the Science of Fish*, 22:526–530, 1975.

[31] D. Rogers and T. Tanimoto. A computer program for classifying plants. *Science*, 21:1115–1118, 1960.

[32] E. Rogot and I.D. Goldberg. A proposed index for measuring agreement in test-retest studies. *Journal of Chronic Disease*, 19:991–1006, 1966.

[33] P. Russel and T. Rao. On habitat and association of species of anopheline larvae in South-Eastern Madras. *Journal of the Malaria Institute of India*, 3(1):153–178, 1940.

[34] R. Santelices, J.A. Jones, Y. Yu, and M.J. Harrold. Lightweight fault-localization using multiple coverage types. In *Proceedings of the International Conference on Software Engineering*, pp. 56–66, 2009.

[35] W.A. Scott. Reliability of content analysis: the case of nominal scale coding. *Public Opinion Quart*, 19:321–325, 1955.

[36] M. Weiser. Program slicing. In *Proceedings of the International Conference on Software Engineering*, pp. 439–449, 1981.

[37] E. Wong, Y. Qi, L. Zhao, and K. Cai. Effective fault localization using code coverage. In *Proceedings of the Annual International Computer Software and Applications Conference*, pp. 449–456, 2007.

[38] X. Xie, T.Y. Chen, F.C. Kuo, and B. Xu. A theoretical analysis of the risk evaluation formulas for spectrum-based fault localization. *ACM Transactions on Software Engineering and Methodology*, 22(4), Article no. 31, 2013.

[39] Y. Yu, J.A. Jones, and M.J. Harrold. An empirical study of the effects of test-suite reduction on fault localization. In *Proceedings of the International Conference on Software Engineering*, pp. 201–210, 2008.

[40] A. Zeller. Isolating cause-effect chains from computer programs. In *Proceedings of the ACM SIGSOFT Symposium on Foundations of Software Engineering*, pp. 1–10, 2002.

[41] X. Zhang, N. Gupta, and R. Gupta. Locating faults through automated predicate switching. In *Proceedings of the International Conference on Software Engineering*, pp. 272–281, 2006.

[42] Z. Zhang, W.K. Chan, T.H. Tse, B. Jiang, and X. Wang. Capturing propagation of infected program states. In *Proceedings of the Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pp. 43–52, 2009.

[43] Z. Zhang, B. Jiang, W.K. Chan, T.H. Tse, X. Wang. Fault localization through evaluation sequences. *Journal of Systems and Software*, 83(2):174–187, 2010.