2015 IEEE 39th Annual International Computers, Software & Applications Conference

A Revisit of a Theoretical Analysis on Spectrum-Based Fault Localization

Tsong Yueh Chen[2], Xiaoyuan Xie[1,*], Fei-Ching Kuo[2], Baowen Xu[3]

[1] State Key Lab of Software Engineering, Wuhan University
Wuhan 430072, China
xxie@whu.edu.cn

[2] Department of Computer Science and Software Engineering, Swinburne University of Technology
Hawthorn, Victoria 3122, Australia
{tychen,dkuo}@swin.edu.au

[3] Department of Computer Science and Technology, Nanjing University
Nanjing 210093, China
bwxu@nju.edu.cn

***Abstract*—As a promising automatic fault localization technique, Spectrum-Based Fault Localization (SBFL) has been proposed and widely studied for years, in which the effectiveness of risk evaluation formula is one of the most popular research topics. We have developed a framework to support the theoretical analysis of risk evaluation formulas, via subset relations. In this paper, we would like to further justify the assumptions used in our framework, which are not only to make the theoretical analysis be feasible, but they are actually required either explicitly or implicitly by SBFL.**

I. Introduction

As a promising automatic fault localization technique, Spectrum-Based Fault Localization (SBFL) has been proposed and widely studied for years, in which the effectiveness of risk evaluation formula is one of the most popular research topics.

In previous studies, people were interested in designing effective risk evaluation formulas [1]–[4] and comparing their performance [3], [5]–[7]. In all these studies, the empirical approach is adopted to investigate and measure the effectiveness of the risk evaluation formulas. However, it is widely known that empirical studies normally have various threats to validity, which can hardly lead to definite conclusions. Therefore, some researchers started to investigated the performance of risk evaluation formulas from a theoretical perspective. For example, Naish et al. [8] first proposed a model-based method to perform a hybrid analysis that contains both theoretical proof and empirical studies. We have also developed a framework to support the theoretical analysis of risk evaluation formulas [9]. In our framework, subset relations are utilized to investigate the relations between two formulas. In [9], [10], 30 manually designed formulas are investigated, in which two groups of equivalent formulas are proved to be maximal. In our follow-up studies, we further investigate 30 formulas evolved via Genetic Programming (GP) and have proved that four of them are maximal [11]. In our most recent study, we have given a complete necessary and sufficient condition of being a maximal formula, and also proved the non-existence of the greatest formula [12].

*Corresponding author.

However, there have been some concerns on the assumptions adopted in our theoretical framework. In this paper, we would like to further justify these assumptions and consequently clarify the significance of our theoretical framework.

II. Background

A. Spectrum-Based Fault Localization (SBFL)

SBFL utilizes testing results and program spectrum to assess the risk value for each program entity. The testing result associated with each test case records whether a test case is *failed* or *passed*. While a program spectrum is a collection of data that provides a specific view on the dynamic behaviour of software. Generally speaking, it records the run-time profiles about various program entities (such as statements, branches, paths, etc.) under a specific test suite. The run-time profile could be the binary coverage status, the number of times that the entity has been covered, and the program state before and after executing the program entity, etc. The most commonly adopted program spectrum is the binary coverage status for program statement, which is also used in this paper.

Given a program $PG=<s_1, s_2, ..., s_n>$ with n statements and executed by test suite $TS=\{t_1, t_2, ..., t_m\}$, testing results and program spectrum are recorded and transformed into vector $A_i=<e_f^i, e_p^i, n_f^i, n_p^i>$ for each statement s_i, where e_f^i and e_p^i represent the number of test cases in *TS* that execute statement s_i and return the testing result of *fail* or *pass*, respectively; while n_f^i and n_p^i denote the number of test cases that do not execute s_i, and return the testing result of *fail* or *pass*, respectively. Then, a risk evaluation formula R is applied on each statement s_i to assign a real value that indicates the risk of being faulty for s_i. A statement with higher risk value is interpreted to have a higher possibility to be faulty, which therefore should be examined with higher priority. Hence, after assigning the risk values to all statements, the statements are sorted descendingly according to their risk values and debugging starts from the top to the bottom of the list.

0730-3157/15 $31.00 © 2015 IEEE
DOI 10.1109/COMPSAC.2015.196

B. Theoretical framework

With the development of more and more risk evaluation formulas, people began to investigate their performance. We have recently developed a theoretical framework to compare the performance between different formulas [9]. Our framework divides a program with n statements into three mutually exclusive sets with respect to the faulty statement, under a particular pair of test suite and risk evaluation formula, as follows.

Definition 1. Given a program with n statements $PG=<s_1, s_2, ..., s_n>$, a test suite of m test cases $TS=\{t_1, t_2, ..., t_m\}$, and a risk evaluation formula R, which assigns a risk value to each program statement. For each statement s_i, a vector $A_i=<e_f^i, e_p^i, n_f^i, n_p^i>$ can be constructed from TS, and $R(s_i)$ is a function of A_i. For any faulty statement s_f, following three subsets are defined.

$$S_B^R = \{s_i \in S | R(s_i) > R(s_f), 1 \leq i \leq n\}$$
$$S_F^R = \{s_i \in S | R(s_i) = R(s_f), 1 \leq i \leq n\}$$
$$S_A^R = \{s_i \in S | R(s_i) < R(s_f), 1 \leq i \leq n\}$$

Our frameworks adopts the most commonly used effectiveness measurement, namely, Expense metric, which is the percentage of code that needs to be examined before the faulty statement is identified [13]. Obviously, a lower Expense of formula R indicates a better performance. Let R_1 and R_2 be two risk evaluation formulas and E_1 and E_2 denote the Expenses with respect to the same faulty statement for R_1 and R_2, respectively. The framework defines two types of relations between R_1 and R_2 as follows.

Definition 2 (Better)**.** R_1 is said to be *better* than R_2 (denoted as $R_1 \rightarrow R_2$) if for any program, faulty statement s_f, test suite and consistent tie-breaking scheme, we have $E_1 \leq E_2$.

Definition 3 (Equivalent)**.** R_1 and R_2 are said to be *equivalent* (denoted as $R_1 \leftrightarrow R_2$), if for any program, faulty statement s_f, test suite and consistent tie-breaking scheme, we have $E_1 = E_2$.

Definitions 2 and 3 require the adoption of a consistent tie-breaking scheme, which is defined as follows.

Definition 4 (Consistent tie-breaking scheme)**.** Given any two sets of statements S_1 and S_2, which contain elements having the same risk values. A tie-breaking scheme returns the ordered statement lists O_1 and O_2 for S_1 and S_2, respectively. The tie-breaking scheme is said to be consistent, if all elements common to S_1 and S_2 have the same relative order in O_1 and O_2.

This framework provides some interesting relations between two formulas via analysing some subset relations.

Theorem 5. Given any two risk evaluation formulas R_1 and R_2, if for any program, faulty statement s_f and test suite, we have $S_B^{R_1} \subseteq S_B^{R_2}$ and $S_A^{R_2} \subseteq S_A^{R_1}$, then $R_1 \rightarrow R_2$.

Theorem 6. Let R_1 and R_2 be two risk evaluation formulas. If we have $S_B^{R_1} = S_B^{R_2}$, $S_F^{R_1} = S_F^{R_2}$ and $S_A^{R_1} = S_A^{R_2}$ for any program, faulty statement s_f and test suite, then $R_1 \leftrightarrow R_2$.

With Theorem 6, six equivalent groups of formulas (namely, ER1 to ER6) are identified from 30 commonly adopted formulas [9]. The detailed and complete proofs that formulas within each group share the same subset division, can be found in [10]. Actually, as stated in [9], Naish et al. [8] have also investigated the six equivalent groups, but they have used the conditions that all statements (for ER2, ER3, ER4 and ER6) or part of the statements (for ER1 and ER5) to produce identical rankings to prove the same ranking of s_f for each group. Obviously, these conditions are special cases of ours that are more general by using subset relation and hence without requiring identical ranking within each subset.

III. Empirical study *v.s.* theoretical analysis

Recently, there have been some concerns about whether the theoretical analysis is too ideal to be useful in practice. In this section, we will discuss about them in details.

A. 100% coverage and omission fault

Some studies have argued that the assumption of 100% coverage in the above theoretical framework is not realistic in real-life [14], [15]. We totally agree that in practice it is very common that a test suite cannot achieve a 100% coverage on the whole program. Actually, we have clearly explained in Section 6 Discussion of [9] that our theoretical framework **can be easily** applied in scenarios where the 100% coverage on the whole program is not achieved. In this section, we will further elaborate our arguments from the nature of SBFL.

As a coverage-based debugging technique, SBFL utilizes coverage profiles and testing results to perform a risk assessment, with intuitions of "higher e_f/lower e_p should lead to higher risk value". In other words, SBFL is actually designed for locating non-omission fault, that is, its responsibility is to find the root faulty statement, of which the execution will trigger failure. This is also explicitly indicated in "the assumptions of SBFL" by Steimann et al. [15] - "Every failed test case executes at least one fault whose execution causes the failure". Obviously, this implies that only the risk assessment on those **covered** statements is meaningful, because statements that are **not** covered by **any** test case can **never** be the root non-omission fault of the observed failures. As a consequence, it is unreasonable to investigate statements that are not covered by any test case. This is supported by Steimann et al. [15] - "However, searching the explanation for failed test cases in code that is not covered by any test case, or even not covered by any failed test case, not only makes little sense per se".

In our theoretical analysis, we have explicitly set the scope of fault type as non-omission ones and assumed a 100% coverage. However, this assumption is **never** equivalent to "having a test suite that can achieve a 100% coverage on the **whole** program". Instead, the 100% coverage means that the risk assessment and theoretical analysis are only performed on the covered statements; while those statements without being covered by any test case are excluded from consideration. It should be very clear that our framework is applicable to **any** test suite with **any** coverage level, because we can simply focus on the **covered subset** of the given program, for which the test suite effectively achieves 100% coverage, regardless what level of coverage it really achieves for the whole program. Since the above intuitions and principles are the basis of SBFL, it should be always followed by both theoretical and empirical studies.

Now, let us revisit the two commonly adopted methods in dealing with non-100% coverage in empirical studies:

- Method A: Exclude the non-covered statements, and focus on the covered ones for SBFL.
- Method B: Simply calculate the risk values for an uncovered statements in the same way as we do for an covered statement, by ignoring the coverage level.

It is obvious that Method A complies with the SBFL's nature and should be adopted in reasonable studies. This is consistent with our theoretical assumption and has been suggested by Steimann et al. [15]. On the other hand, Method B may introduce noises [15]. It may not affect some formulas with which the risk values of uncovered statements are no higher than those of the covered statements, especially lower than those of the statements with $e_f^i{=}F$, such as Jaccard. However, for some formulas, it is possible that the risk values of uncovered statements are higher than those of the covered statements, for example, Wong2 ($e_f{-}e_p$). In the latter formulas, noises from the uncovered statements may affect the performance. Actually, things may be even more complicated in some formulas, such as Ochiai, Tarantula, uncovered statements become undefined due to the zero-denominator. Then, how to define these statements will affect the performance of the formula. Two extreme cases can be: (i) rank these statements at the top of the lists; or (ii) rank them at the bottom. Obviously, these two strategies may give totally different results; while the latter one gives better performance and is essentially the same as the above Method A. (Note that it provides the same absolute ranking for the faulty statement as Method A, but lower *Expense* score because more statements are considered and hence a larger denominator. But the comparison results between formulas are the same as the ones with Method A.)

Unfortunately, we usually cannot tell which method an experiment has taken, since it is unnecessary to report such low-level technical details in a paper. And for those who use Method B, we have no way to know how they define the uncovered statements for formulas like Ochiai or Tarantula. Anyway, by adding additional definitions to these formulas, they are no longer the original versions, and the comparison is actually done on those newly defined formulas Ochiai' and Tarantula', rather than their original versions. Moreover, since no details have been reported, how can we guarantee that different people use the same new definition? If different new definitions are used, they are actually investigating different formulas, and hence the results cannot be put together for cross-validation. As mentioned above some empirical studies have suggested to use Method A [15] to avoid the potential noises. Steimann et al. [15] also suggested to further exclude the statements with $e_f{=}0$, which has actually been suggested earlier by Xie et al. [9], [16].

To summarize, the assumptions about the coverage and omission faults are far more than reasonable and should actually be adopted in both empirical and theoretical analysis. Relevant discussion has usually been omitted in empirical studies. However, as a theoretical analysis, we must list them out, to provide precise proofs and definite answers.

B. Tie-breaking scheme

Another concern on the theoretical analysis is about the "consistent tie-breaking scheme".

It is very common that SBFL assigns the same risk values to different statements. In such a case, a tie-breaking scheme is required to further distinguish these statements. Therefore, given a program and a test suite, the Expense is co-determined by formula R and the adopted tie-breaking scheme. In other words, a tie-breaking scheme is **not** a component of a risk evaluation formula. Instead, it is a component of a SBFL technique. Thus, for a fair comparison between various risk evaluation formulas R, the tie-breaking scheme should be the same across different R, which must therefore be applicable to any formula. For example, it is obviously not meaningful to bind formula R_1 with tie-breaking scheme B_1 while bind R_2 with a different tie-breaking scheme B_2, and then treat the comparison between $R_1{+}B_1$ and $R_2{+}B_2$ equally as the comparison between R_1 and R_2.

Therefore, in our theoretical analysis, we require different formulas to use a common tie-breaking scheme. Moreover, we have chosen a general family of tie-breaking schemes, namely, consistent tie-breaking scheme, as a representative in our theoretical analysis. We use it because of the following reasons.

- Instead of being one particular single scheme, consistent tie-breaking scheme actually covers **a large family** of schemes. Based on Definition 4, as long as a tie-breaking scheme always gives the same relative order to any set of tied statements, it is said to be consistent. In other words, such a tie-breaking scheme **does not** care about what particular order should be used for tied statements in one formula. Instead, it only requires the same relative order for the common set of tied statements in different formulas. There can be a wide variety of relative orders, such as the sequential order in the source code or control flow graph, order decided by other information, user defined order based on their debugging habits or experience, or even a fixed random order [1]. Obviously, all of these particular schemes can be practically applied in real-life, of course in empirical studies as well.
- Consistent tie-breaking scheme that always provides identical relative order to tied statements is the most straightforward and intuitive way to be totally independent of the risk formulas. With such a scheme, any noises from the tie-breaking schemes can be excluded such that a fair comparison between formulas can be guaranteed.

As a reminder, choosing the consistent tie-breaking scheme in our theoretical framework does **not** mean that our framework is inapplicable with other tie-breaking schemes. In fact, in our following discussion, we will prove that our propositions in [9] still hold with other commonly adopted tie-breaking schemes, such as "Best", "Worst" and "Average". However, we must also point out that strictly speaking, such tie-breaking schemes are **not realistic** solutions when performing SBFL in practice. Instead, they are just laboratory strategies to estimate the performance of SBFL technique. That is, such tie-breaking schemes cannot be applied in practice simply because we do not know which statement is faulty.

Theorem 7. If R_1 and R_2 satisfy the corresponding subset relations required by Theorem 5 or 6, then by adopting "**Best**

[1] Same random order in different formulas

tie-breaking scheme", we still have $R_1 \rightarrow R_2$ and $R_1 \leftrightarrow R_2$, respectively, for any program and test suite.

Proof: "Best tie-breaking scheme" ranks s_f at the top among all statements in S_F^R. Thus, have $E{=}|S_B^R|{+}1$. Let us denote the difference between E_1 and E_2 as $\delta{=}E_1{-}E_2$. It follows immediately that $\delta{=}|S_B^{R_1}|{-}|S_B^{R_2}|$.

1) If $S_B^{R_1}{\subseteq}S_B^{R_2}$ and $S_A^{R_2}{\subseteq}S_A^{R_1}$, we have $|S_B^{R_1}|{-}|S_B^{R_2}|{\leq}0$. As a consequence, $\delta{\leq}0$, that is, $E_1{\leq}E_2$.
2) If $S_B^{R_1}{=}S_B^{R_2}$ and $S_A^{R_2}{=}S_A^{R_1}$, we have $|S_B^{R_1}|{-}|S_B^{R_2}|{=}0$. As a consequence, $\delta{=}0$, that is, $E_1{=}E_2$.

After Definitions 2 and 3, the theorem is proved. ■

Theorem 8. If R_1 and R_2 satisfy the corresponding subset relations required by Theorem 5 or 6, then by adopting "Worst tie-breaking scheme", we still have $R_1 \rightarrow R_2$ and $R_1 \leftrightarrow R_2$, respectively, for any program and test suite.

Proof: "Worst tie-breaking scheme" ranks s_f at the bottom among all statements in S_F^R. Thus, we have

$$E{=}|S_B^R|{+}|S_F^R|{=}|S|{-}|S_A^R|$$

Let us denote $\delta{=}E_1{-}E_2$, then we have $\delta{=}|S_A^{R_2}|{-}|S_A^{R_1}|$.

1) If $S_B^{R_1}{\subseteq}S_B^{R_2}$ and $S_A^{R_2}{\subseteq}S_A^{R_1}$, we have $|S_A^{R_2}|{-}|S_A^{R_1}|{\leq}0$. As a consequence, $\delta{\leq}0$, that is, $E_1{\leq}E_2$.
2) If $S_B^{R_1}{=}S_B^{R_2}$ and $S_A^{R_2}{=}S_A^{R_1}$, we have $|S_A^{R_2}|{-}|S_A^{R_1}|{=}0$. As a consequence, $\delta{=}0$, that is, $E_1{=}E_2$.

After Definitions 2 and 3, the theorem is proved. ■

Actually, conclusions in the above two theorems have also been discussed in [17]. As a reminder, it can be found from the above proofs that, with "Best" or "Worst" tie-breaking schemes, comparing two formulas does not require the subset relations for both S_B^R and S_A^R. With "Best" tie-breaking scheme, $S_B^{R_1}{\subseteq}S_B^{R_2}$ implies $E_1{\leq}E_2$, and $S_B^{R_1}{=}S_B^{R_2}$ implies $E_1{=}E_2$, regardless of the relations for the other two subsets. While with "Worst" tie-breaking scheme, $S_A^{R_2}{\subseteq}S_A^{R_1}$implies $E_1{\leq}E_2$, and $S_A^{R_1}{=}S_B^{R_2}$ implies $E_1{=}E_2$, regardless of the relations for the other two subsets. However, even though the adoption of these two tie-breaking schemes can simplify the theorem, they were not included in [9] because neither "Best" nor "Worst" is applicable in real life.

Theorem 9. If R_1 and R_2 satisfy $S_B^{R_1}{\subseteq}S_B^{R_2}$ and $S_A^{R_2}{\subseteq}S_A^{R_1}$, then by adopting "Average" tie-breaking scheme, we still have $R_1 \rightarrow R_2$, for any program and test suite.

Proof: "Average" tie-breaking scheme ranks s_f at the medium position among all statements in S_F^R. If $|S_F^R|$ is odd, we always have $E{=}|S_B^R|{+}\frac{|S_F^R|+1}{2}$. If $|S_F^R|$ is even, there are two possible definitions of the medium position, and correspondingly, we have (i) $E{=}|S_B^R|{+}\frac{|S_F^R|}{2}$; or (ii) $E{=}|S_B^R|{+}\frac{|S_F^R|}{2}{+}1$. Since the proofs with definitions (i) and (ii) are very similar, due to the page limitation, we will adopt (i) for even $|S_F^R|$.

Let us denote $\delta{=}E_1{-}E_2$, $\delta_B{=}|S_B^{R_1}|{-}|S_B^{R_2}|$ and $\delta_A{=}|S_A^{R_1}|{-}|S_A^{R_2}|$. Given R_1 and R_2, since $S_B^{R_1}{\subseteq}S_B^{R_2}$ and $S_A^{R_2}{\subseteq}S_A^{R_1}$, we have $\delta_B{=}|S_B^{R_1}|{-}|S_B^{R_2}|{\leq}0$ and $\delta_A{=}|S_A^{R_1}|{-}|S_A^{R_2}|{\geq}0$. Then,

$$|S|{=}|S_B^{R_1}|{+}|S_F^{R_1}|{+}|S_A^{R_1}|{=}|S_B^{R_2}|{+}|S_F^{R_2}|{+}|S_A^{R_2}|$$

can be re-written as

$$(|S_B^{R_1}|{-}|S_B^{R_2}|){+}(|S_A^{R_1}|{-}|S_A^{R_2}|){=}\delta_B{+}\delta_A{=}|S_F^{R_2}|{-}|S_F^{R_1}| \quad (1)$$

There are following possible cases.

1) Consider the case that both $S_F^{R_1}$ and $S_F^{R_2}$ are even or odd. Then, we have $\delta{=}(|S_B^{R_1}|{+}\frac{|S_F^{R_1}|}{2}){-}(|S_B^{R_2}|{+}\frac{|S_F^{R_2}|}{2})$. After Equation (1), we have $\delta{=}\frac{1}{2}(\delta_B{-}\delta_A)$. Since $\delta_B{\leq}0$ and $\delta_A{\geq}0$, we have $\delta{\leq}0$. In other words, we have $E_1{\leq}E_2$.
2) Consider the case that $S_F^{R_1}$ is odd while $S_F^{R_2}$ is even. Then, we have $\delta{=}(|S_B^{R_1}|{+}\frac{|S_F^{R_1}|+1}{2}){-}(|S_B^{R_2}|{+}\frac{|S_F^{R_2}|}{2})$. After Equation (1), we have $\delta{=}\frac{1}{2}(\delta_B{-}\delta_A{+}1)$. In this case, since $S_F^{R_1}$ is odd while $S_F^{R_2}$ is even, then δ_B and δ_A cannot be 0 at the same time. Thus, we must have $\delta_B{-}\delta_A{\leq}{-}1$ and hence $\delta{\leq}0$. In other words, we also have $E_1{\leq}E_2$.
3) Consider the case that $S_F^{R_1}$ is even while $S_F^{R_2}$ is odd. Then, we have $\delta{=}(|S_B^{R_1}|{+}\frac{|S_F^{R_1}|}{2}){-}(|S_B^{R_2}|{+}\frac{|S_F^{R_2}|+1}{2})$. After Equation (1), we have $\delta{=}\frac{1}{2}(\delta_B{-}\delta_A{-}1)$. Similar to the second case, δ_B and δ_A cannot be 0 at the same time. Thus, we must have $\delta_B{-}\delta_A{\leq}{-}1$ and hence $\delta{<}0$. In other words, we have $E_1{<}E_2$.

In summary, if $S_B^{R_1}{\subseteq}S_B^{R_2}$ and $S_A^{R_2}{\subseteq}S_A^{R_1}$, then by adopting "Average" tie-breaking scheme, we also have $E_1{\leq}E_2$, for any program and test suite. After Definition 2, the theorem is proved. ■

Theorem 10. If R_1 and R_2 satisfy $S_B^{R_1}{=}S_B^{R_2}$, $S_F^{R_1}{=}S_F^{R_2}$ and $S_A^{R_2}{=}S_A^{R_1}$, then by adopting any "Average tie-breaking scheme", we still have $R_1 \leftrightarrow R_2$, for any program and test suite.

Proof: Since $S_B^{R_1}{=}S_B^{R_2}$, $S_F^{R_1}{=}S_F^{R_2}$ and $S_A^{R_2}{=}S_A^{R_1}$, then we have $\delta_B{=}0$, $\delta_A{=}0$, and $S_F^{R_1}$ and $S_F^{R_2}$ must both be either even or odd. Thus, we have $\delta{=}\frac{1}{2}(\delta_B{-}\delta_A){=}0$, which means $E_1{=}E_2$. After Definition 3, the theorem is proved. ■

C. Multiple faults

As discussed above, SBFL formulas are designed based on intuition of "higher e_f/lower e_p should lead to higher risk value". It is not difficult to find out that such an intuition is only meaningful for single (non-omission) fault scenario [18]. Fortunately, DiGiuseppe and Jones [18] also have demonstrated that "in terms of localizing at least one, most prominent, fault, the performance of SBFL is not adversely affected by the increasing number of faults, even in the presence of fault localization interference". Such an evidence implies that the conclusions of our theoretical analysis are equally useful for multiple faults.

On the other hand, an attractive idea was proposed to assist the application of SBFL in multiple-fault scenario, namely, parallel debugging approaches [19]–[21]. In these studies, test cases are first clustered into several specialized test suites based on various execution information, and each of the test

suites targets an individual single fault. In practice, each specialized test suite is dispatched to a particular debugger, who is supposed to focus on the corresponding single fault. In other words, by properly clustering the test suite, the multiple-fault scenario can be transformed into single-fault scenario, in which our theoretical framework can be applied. For more detailed discussion, please refer to the section of Discussion in [9].

D. Some plausible causes for the inconsistence between empirical and theoretical analyses

Most of the experimental studies have shown consistent results with the theoretical analysis [3], [6]–[8], [22], [23]. However, there exist some studies which show inconsistence with the theoretical analysis [14], [24], [25]. Some plausible causes for the inconsistence are as follows.

1) The experimental setup varies in different formula investigations. For example, Yu et al. [24] and Jiang et al. [25] and have used different tie-breaking schemes for Tarantula and CBI.
2) Existence of noise of $e_f^f<F$ for faulty statement s_f. As discussed above, based on the nature of SBFL, for single non-omission fault, we should have $e_f^f=F$. However, such a situation may not be observed in experiments. Note that for some crash failures, the coverage information may not be properly collected. In such cases, the program crashes when the execution reaches the faulty statement, and the coverage collector also stops working at the same time and fails to properly record the coverage of the faulty statement. Therefore, it is possible to have $e_f^f<F$, which violates the nature of SBFL. Thus, this should be considered as a type of noises, and it is actually a very common cause why the empirical results violate the theoretical analysis. In our most recent studies [12], we have proved that "assigning higher risk values to statements with $e_f^i=F$ than to those with $e_f^i<F$" is the necessary and sufficient condition for formulas of being maximal (intuitively speaking, these formulas are good because they assign highest risk values to the single non-omission faulty statement that always have $e_f^f=F$). In other words, maximal formulas are only interested in those "possible faulty statements whose $e_f^i=F$". Some maximal formulas even ignore those statements with $e_f^i<F$", such as Naish1 that assigns -1 to all statements with $e_f^i<F$". However, if the noise of $e_f^f<F$ is introduced, the faulty statement will not be the focus of the maximal formula and then the performance of the formula decreases. For example, Naish1 is very sensitive to this type of noises, with which the performance of Naish1 significantly decreases.
3) Existence of noise from omission faults. Some people use Siemens Suite that contains omission faults, and different people use different strategies to handle this case, by considering the preceding or succeeding statement of the missing statement as the "faulty statement". Furthermore, the "preceding or succeeding" statement may have different interpretations, such as "the line order of source code" or "the order according to the control-flow graph". And there is no consensus on the interpretation. Besides, such omission fault may result in $e_f^f<F$.

Moreover, it is possible to have inconsistence between different empirical results, due to the above reasons (internal threats to validities in experimental studies), as well as the non-exhaustiveness of experimental analysis. For example, Le al et. [14] observes Ochiai outperforms Naish2 and Naish 1, which is contradictory to the experimental results in [8] (Table XI). Without any theoretical analysis, it is impossible to identify whether the inconsistency is because there is actually no relation between the these formulas (i.e. $R_1 \nleftrightarrow R_2$) or because there are noises in either experiments. But from theoretical analysis, we can have the answer. Actually, we have precisely defined the assumptions and the scope of our theoretical framework. Since all proofs of the theoretical analysis are explicitly stated, an independent verification for the results of the theoretical analysis can be easily conducted. However, it is a common practice to not provide these technical details in empirical studies, and hence it is extremely difficult, if not impossible, to verify whether an experimental analysis has been conducted without any mistake.

IV. Conclusion

In summary, the assumptions in [9] are not only to make the theoretical analysis be feasible, but they are actually required either explicitly or implicitly by SBFL. In fact, many empirical studies have adopted such assumptions as well. However, we still believe that empirical and theoretical analysis are complementary to each other: comprehensive empirical results usually provide hints for theoretical hypotheses; while theoretical conclusions will help to lay a solid foundation for future experimental studies.

Acknowledgment

This work is supported by an ARC Discovery Project (DP120104773).

References

[1] J. A. Jones, M. J. Harrold, and J. Stasko, "Visualization of test information to assist fault localization," in *Proceedings of the 24th International Conference on Software Engineering*, Florida, USA, 2002, pp. 467–477.

[2] M. Chen, E. Kiciman, E. Fratkin, A. Fox, and E. Brewer, "Pinpoint: problem determination in large, dynamic internet services," in *Proceedings of the 32th IEEE/IFIP International Conference on Dependable Systems and Networks*, Washington DC, USA, 2002, pp. 595–604.

[3] R. Abreu, P. Zoeteweij, and A. J. C. van Gemund, "An evaluation of similarity coefficients for software fault localization," in *Proceedings of the 12th Pacific Rim International Symposium on Dependable Computing*, Riverside, USA, 2006, pp. 39–46.

[4] W. E. Wong, V. Debroy, and B. Choi, "A family of code coverage-based heuristics for effective fault localization," *Journal of Systems and Software*, vol. 83, no. 2, pp. 188–208, 2010.

[5] J. A. Jones and M. J. Harrold, "Empirical evaluation of the tarantula automatic fault-localization technique," in *Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering*, Long Beach, USA, 2005, pp. 273–282.

[6] R. Abreu, P. Zoeteweij, and A. J. C. van Gemund, "On the accuracy of spectrum-based fault localization," in *Proceedings of Testing: Academic and Industrial Conference Practice and Research Techniques-MUTATION*, Windsor, UK, 2007, pp. 89–98.

[7] 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*, vol. 82, no. 11, pp. 1780–1792, 2009.

[8] L. Naish, H. J. Lee, and K. Ramamohanarao, "A model for spectra-based software diagnosis," *ACM Transactions on Software Engineering and Methodology*, vol. 20, no. 3, pp. 11:1–11:32, 2011.

[9] X. Xie, T. Y. Chen, F.-C. Kuo, and Xu, "A theoretical analysis of the risk evaluation formulas for spectrum-based fault localization," *ACM Transactions on Software Engineering and Methodology*, vol. 22, no. 4, pp. 31:1–31:40, 2013.

[10] X. Xie, "On the analysis of spectrum-based fault localization," Ph.D. dissertation, Swinburne University of Technology, May 2012.

[11] X. Xie, F.-C. Kuo, T. Y. Chen, S. Yoo, and M. Harman, "Provably optimal and human-competitive results in SBSE for spectrum based fault localisation," in *Proceedings of the 5th Symposium on Search Based Software Engineering (SSBSE)*, Saint Petersburg, Russia, 2013, pp. 224–238.

[12] S. Yoo, X. Xie, F.-C. Kuo, T. Y. Chen, and M. Harman, "No Pot of Gold at the End of Program Spectrum Rainbow: Greatest Risk Evaluation Formula Does Not Exist," Department of Computer Science, University College London, Tech. Rep. RN/14/14, 2014.

[13] S. Yoo, "Evolving human competitive spectra-based fault localisation techniques," in *Proceedings of the 4th International Symposium on Search-Based Software Engineering*, Trento, Italy, 2012, pp. 244–258.

[14] T.-D. Le, F. Thung, and D. Lo, "Theory and Practice, Do They Match? A Case with Spectrum-Based Fault Localization," in *Proceedings of the 29th IEEE International Conference on Software Maintenance*, Eindhoven, Netherlands., 2013, pp. 380–383.

[15] F. Steimann, M. Frenkel, and R. Abreu, "Threats to the Validity and Value of Empirical Assessments of the Accuracy of Coverage-based Fault Locators," in *Proceedings of the 2013 International Symposium on Software Testing and Analysis*, NY, USA, 2013, pp. 314–324.

[16] X. Xie, T. Y. Chen, and B. W. Xu, "Isolating suspiciousness from spectrum-based fault localization techniques," in *Proceedings of the 10th International Conference on Quality Software*, Zhangjiajie, China, 2010, pp. 385–392.

[17] W. Wong, V. Debroy, R. Gao, and Y. Li, "The DStar Method for Effective Software Fault Localization," *IEEE Transactions on Reliability*, vol. 63, no. 1, pp. 290–308, March 2014.

[18] N. DiGiuseppe and J. A. Jones, "On the influence of multiple faults on coverage-based fault localization," in *Proceedings of the International Symposium on Software Testing and Analysis*, Toronto, Canada, 2011, pp. 199–209.

[19] C. Liu and J. Han, "Failure proximity: a fault localization-based approach," in *Proceedings of the 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, New York, USA, 2006, pp. 46–56.

[20] A. X. Zheng, M. I. Jordan, B. Liblit, M. Naik, and A. Aiken, "Statistical debugging: simultaneous identification of multiple bugs," in *Proceedings of the 23rd International Conference on Machine Learning*, New York, USA, 2006, pp. 1105–1112.

[21] J. A. Jones, J. F. Bowring, and M. J. Harrold, "Debugging in parallel," in *Proceedings of the International Symposium on Software Testing and Analysis*, New York, USA, 2007, pp. 16–26.

[22] R. Santelices, J. A. Jones, Y. Yu, and M. J. Harrold, "Lightweight fault-localization using multiple coverage types," in *Proceedings of the 31st International Conference on Software Engineering*, Vancouver, Canada, 2009, pp. 56–66.

[23] H. J. Lee, L. Naish, and K. Ramamohanarao, "The effectiveness of using non redundant test cases with program spectra for bug localization," in *Proceedings of the 2nd IEEE International Conference on Computer Science and Information Technology*, Beijing, China, 2009, pp. 127–134.

[24] 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 30th International Conference on Software Engineering*, Leipzig, Germany, 2008, pp. 201–210.

[25] B. Jiang, Z. Zhang, T. H. Tse, and T. Y. Chen, "How well do test case prioritization techniques support statistical fault localization," in *Proceedings of the 33rd Annual International Conference on Computer Software and Applications*, vol. 1, Seattle, USA, 2009, pp. 99–106.