Comprehensive Evaluation of Association Measures for Fault Localization

Lucia, David Lo, Lingxiao Jiang, Aditya Budi
School of Information Systems, Singapore Management University
Emails: {lucia.2009,davidlo,lxjiang,adityabudi}@smu.edu.sg

***Abstract*—In statistics and data mining communities, there have been many measures proposed to gauge the strength of association between two variables of interest, such as odds ratio, confidence, Yule-Y, Yule-Q, Kappa, and gini index. These association measures have been used in various domains, for example, to evaluate whether a particular medical practice is associated positively to a cure of a disease or whether a particular marketing strategy is associated positively to an increase in revenue, etc. This paper models the problem of locating faults as association between the execution or non-execution of particular program elements with failures. There have been special measures, termed as *suspiciousness measures*, proposed for the task. Two state-of-the-art measures are Tarantula and Ochiai, which are different from many other statistical measures. To the best of our knowledge, there is no study that comprehensively investigates the effectiveness of various association measures in localizing faults. This paper fills in the gap by evaluating 20 well-known association measures and compares their effectiveness in fault localization tasks with Tarantula and Ochiai. Evaluation on the Siemens programs show that a number of association measures perform statistically comparable as Tarantula and Ochiai.**

I. Introduction

Software debugging is a difficult and expensive activity to perform. When a program failure occurs in a program execution, the execution traces may be long and contain failure-irrelevant information. Locating the root cause of the failure, which may be far away from the failure point, is non-trivial. Often, full execution traces are not even available for debugging if programs fail in the field. US National Institute of Standards and Technology in 2002 has stated that software bugs cost US economy 59.5 billion dollars annually [46], and testing and debugging activities account for 30 to 90 percent of labor expended for a project [8].

Many approaches have been proposed to help in automating debugging processes, especially in localizing root causes of failures (i.e., *faults*) in programs [20], [27], [29], [31], [33], [34], [36], [38], [51]. One common kind of approaches is spectrum-based fault localization [1], [11], [14], [27], [31], where program traces or abstractions of traces (called *program spectra*) are used to represent program behaviors and the spectra of correct executions and failed executions are compared. The comparison often employs statistical analysis, and program elements (e.g., statements, basic blocks, functions, components, etc.) that are observed more often in failed executions than in correct executions (or statistically correlate with failures) are located and deemed as suspicious and then presented to developers.

Spectrum-based fault localization techniques are promising as they are often lightweight and have good accuracy. Among spectrum-based fault localization techniques, two state-of-the-art techniques are Tarantula [26], [27] and Ochiai [1]–[3]. Both approaches propose suspiciousness measures that aim to *associate the execution of a particular program element with program failures*. A suspiciousness score is inferred for each program element based on the suspiciousness measures, and program elements are ranked and investigated for faults according to their suspiciousness scores.

We observe that a certain program failure could be associated with a non-execution of a particular program element. A bug in the controlling program element (e.g. wrong condition in an `if` statement) could cause the non-execution of a particular program element that will induce the program failure. Therefore, we would like to model the association, as the following question:

> *What is the association between the execution or non-execution of a particular program element with the occurrences of a failure?*

Based on this view, a general solution for fault localization can naturally emerge: measure the association of program elements with failures and the stronger association an element has with failures the more likely it is a fault.

Tarantula and Ochiai have been used as measures for fault localization. In the data mining and statistics community, there are various association measures, such as odds ratio [6], Yule Q [47], and Yule Y [48]. They are proposed to measure the strength of association of two variables. For example, one might be interested in the association between the application of a particular medical treatment with the recovery from an illness, or in the association between a business strategy with the revenue change. Unfortunately, there has not been any study that investigates the effectiveness of the rich varieties of association measures for fault localization.

This paper aims to fill this research gap by investigating the effectiveness of 20 popular association measures for the purpose of fault localization and comparing them with Tarantula and Ochiai. In particular, we are interested in answering the following research questions (RQs):

- **RQ 1.** Are vanilla or off-the-shelf association measures accurate enough in localizing faults?
- **RQ 2.** Which of the off-the-shelf association measures provide better accuracy for fault localization?

RQ 3. What is the relative performance of the off-the-shelf association measures as compared to well-known suspiciousness measures for fault localization, Tarantula and Ochiai in particular?

To answer the above questions, we investigate and compare the accuracies of Tarantula, Ochiai, and the additional 20 association measures on the standard Siemens test program suite obtained from the Aristotle Analysis System and Software-artifact Infrastructure Repository [16], [41]. The Siemens suite comes with seven programs seeded with bugs, a test oracle to decide between failures and non-failures, and a set of test cases. We compute various accuracy metrics to evaluate the effectiveness of the association measures in localizing faults to answer the above research questions. We show that a number of association measures have similar performance as Tarantula and Ochiai.

The contributions of this work are as follows:

1) We model the problem of fault localization as the association between the execution or non-execution of program elements and failures.
2) We comprehensively investigate the effectiveness of a number of association measures on fault localization.
3) We highlight a few promising association measures with statically comparable performance as Tarantula and Ochiai.
4) We provide a partial order of association measures in terms of their accuracies for fault localization.

The structure of this paper is as follows. Section II discusses related work. Section III discusses the fundamental concepts of spectrum-based fault localization and association measures. Section IV discusses the particular association measures considered in the paper. Section V describes our empricial evaluation and comparison of the association measures. Finally, we conclude and discuss future work in Section VI.

II. Related Work

In this section, we describe closely related studies on fault localization and association measures. The survey here is by no means a complete list of all related studies.

A. Fault Localization

Recently, there are a lot of studies on fault localization and automated debugging. There are even different ways to categorize these studies. Based on the data used in the approaches, fault localization techniques can be classified into *spectrum-based* and *model-based*. Spectrum-based fault localization techniques often use program spectra, which are program traces or abstractions of program traces that represent program runtime behaviors in certain ways, to correlate program elements (e.g., statements, basic blocks, functions, and components) with program failures (often with the help of statistical analysis).

Many spectrum-based fault localization techniques [13], [14], [27], [30], [38], [50] take as inputs two sets of spectra, one for successful executions and the other for failed executions, and report candidate locations where root causes of program failures (i.e., faults) may reside. Given a failed program spectrum and a set of correct spectra, Renieris and Reiss present a fault localization tool WHITHER [38] that compares the failed execution to the nearest correct execution and reports the most suspicious locations in the program. Zeller and Hildebrandt propose a technique called Delta Debugging that can simplify and isolate failure-inducing inputs in a binary-search-like fashion [49], and Zeller applies Delta Debugging to search for the minimum state differences between a failed execution and a successful execution that may cause the failure [50]. Cleve and Zeller also extended the work by incorporating a search capability for cause transitions, namely locations in the program where new relevant variables become a failure cause, in their tool called AskIgor [13]. Liblit et al. propose a technique to search for predicates whose true evaluation correlates with failures [30]. Chao et al. extend the work by incorporating information on the outcomes of multiple predicate evaluations in a program run in their tool called SOBER [14]. All of these techniques need to compare spectra of failed executions with those of successful executions in some way. Evaluating the effectiveness of various association measures can complement all of these techniques by helping to locate the most failure-relevant program elements quickly and improving their performance.

Other spectrum-based techniques [20], [25], [43], [52] only use failed executions as the input and systematically alter the program structure or program runtime states to locate faults. Zhang et al. [52] search for faulty program predicates by switching the states of program predicates at runtime. Sterling and Olsson use the concept of program chipping [43] to automatically remove parts of a program so that the part that contributes to the failure may become more apparent. While their tool, ChipperJ, works on syntax trees for Java programs, Gupta et al. [20] work on program dependency graphs and use the intersection of forward and backward program slices to reduce the sizes of failure-relevant code for further inspection. Jeffrey et al. use a value profile based approach to rank program statements according to their likelihood of being faulty [25]. These fault localization techniques do not compare the spectra of failed executions with those of successful executions, and association measures are generally not applicable to them.

Compared with spectrum-based techniques, model-based debugging techniques [17], [34] are often more accurate, but heavyweight since they are based on more expensive logic reasoning over formal models of programs. Many static and dynamic analysis techniques [32], [33], [44] can be classified as model-based debugging as well. Although few model-based techniques have employed the concept of failure association, incorporating association measures and other statistical analyses into program models can be a future direction for improving the performance of model-based debugging techniques.

In our study, we focus on comparisons with two state-of-the-art spectrum-based fault localization techniques in the literature (to the best of our knowledge) that can be modeled as the association question, namely Tarantula [26], [27] and

Ochiai [1]–[3]. Our study extends the work by Tarantula and Ochiai in the sense that we systematically apply and evaluate more than 20 association measures and find promising measures for the task of fault localization.

B. Studies On Association Measures

There have been a number of studies proposed in the statistics and data mining community on measures of association between variables since the early 20th century. These include measures such as Yule's Q and Yule's Y [47], [48]. Other measures, such as odds ratio [6], are also commonly considered and utilized in various domains, such as medical [7] and social science [23]. In the data mining community, Agrawal and Srikant have proposed the work on association rule mining which aims to infer associations from two itemsets in a transaction dataset in the early 90's [5]. In that work the metrics of support and confidence for measuring the strength of an association are proposed. Various other metrics, such as interest and collective strength, are proposed later. We describe these measures in detail in Section IV for our evaluations.

Tan et al. investigate various association measures, compare their properties, and outline the benefits and limitations of each from a computation point of view [45]. The measures are revisited by Geng and Hamilton by including measures for aggregated data summaries [18]. In this paper, we extend their work in the specific domain of fault localization by comparing the measures based on their ability to provide high suspiciousness for buggy program elements.

III. Concepts & Definitions

In this section we formally introduce the problem of spectrum-based fault localization as an association question. Next, we describe the concept of dichotomy matrix that will be used to calculate the strength of the association between two variables.

A. Spectrum-Based Fault Localization

This problem starts with a faulty program, a set of test cases, and a test oracle. The set of test cases are run over the faulty program and observations of how the program runs on each of the test cases are recorded as program spectra. A program spectrum [22], [39] is simply a set of data or values collected at runtime; each value could be a program state, or a counter or a flag for a program element. A test oracle is available to label whether a particular output or execution of a test case is correct or wrong. Wrong executions are classified as program failures. The task of a fault localization tool is to find the program elements that are responsible for the failures (i.e., the faults or the root causes) based on the program spectra of both correct and wrong executions.

*Definition 3.1 (***Association Question):** Given a program $P = \{e_1, ..., e_n\}$ and a set of program spectra $T = T_s \cup T_f$ for P, where P is comprised of n elements $e_1, ..., e_n$ and T is comprised of the spectra for correct executions T_s and the spectra for wrong executions T_f, the *association question* is to measure (1) the strength of the association between the *execution of each* e_i and the program failures and (2) the strength of the association between the *non-execution of each* e_i and the program failures, and (3) assign a quantitative score to each e_i to indicate the likelihood that e_i is a fault. We call the score for each e_i the *suspiciousness score*.

Block ID	Program Elements	T15	T16	T17	T18
1	int count; int n; Ele *proc; List *src_queue, *dest_queue; if (prio >= MAXPRIO) /*maxprio=3*/	•	•	•	•
2	{return;}		•	•	•

3	src_queue = prio_queue[prio];
	dest_queue = prio_queue[prio+1];
	count = src_queue->mem_count;
	if (count > 1) /* Bug- supposed : count>0*/ {
		•	•	•	•

4	n = (int) (count*ratio + 1);
	proc = find_nth(src_queue, n);
	if (proc) {
			•	•	

5	src_queue = del_ele(src_queue, proc);
	proc->priority = prio;
	dest_queue = append_ele(dest_queue, proc); } }}
			•	•	

	Pass/Fail of Test Case Execution :	Pass	Pass	Pass	Fail

Fig. 1. Example of block-hit program spectra

There have been various spectra proposed in the literature [1], [22], but we are particularly interested in *block-hit program spectra*, each of which consists of a set of flags to indicate whether each basic block is executed or not in each test case.[1] An example of block-hit program spectra is shown in Figure 1. The second column is a code excerpt from the Siemens test programs; the "Block ID" column indicates the ID of the containing basic block of the statements. A bug lies in the `if` condition in Block 3, causing Blocks 4–5 to be skipped when the variable `count` is 1. The other columns indicate whether each basic block is executed in each of test cases 15, 16, 17, and 18 along with the information whether the test case is pass or fail. In this example, • denotes a basic block is executed by a test case and an empty cell denotes the block is not executed by the test case.

There are also various ways to calculate the association strengths based on various association measures. Evaluating these association measures is the focus of this paper and is presented in the following sections. In addition, transforming association strengths into suspiciousness scores for e_i can be done in different ways. However, the transformation functions involved in this step can often be very simple (e.g., an identity function or a maximum function) as long as the association measures can provide accurate association strengths with failures for each e_i. Separating the calculation of suspiciousness scores from association strengths allows more flexible combinations of the two association strengths for each e_i, but it is not our focus in this purpose to evaluate different combinations.

1) Tarantula: Jones and Harrold propose Tarantula [27] to rank program elements based on their suspiciousness measure. Intuitively, a program element is more suspicious if it appears

[1]We acknowledge that different spectra may have different effects in fault localization. We use *block-hit program spectra* in this paper mainly for establishing the same ground for comparison with Tarantula and Ochiai. An interesting future work will be to evaluate the applicability and effectiveness of various association measures on different kinds of program spectra.

in failed executions more frequently than in correct executions. Now we introduce the following common notations which are used in the association and suspiciousness measures in the rest of the paper: Given a program P and a test suite for P, n is the total number of test cases in the test suite; $n(e)$ is the number of test cases that run through a particular element e in P; n_s is the number of test cases that succeed; n_f is the number of test cases that fail; $n_s(e)$ is the number of test cases that run through a particular element e and succeed; $n_f(e)$ is the number of test cases that run through a particular element e and fail.

Then, the suspiciousness score introduced by Tarantula for each element e can be represented as the following:

$$suspiciousness(e) = \frac{\frac{n_f(e)}{n_f}}{\frac{n_s(e)}{n_s} + \frac{n_f(e)}{n_f}}$$

Based on block-hit program spectra in Figure 1, the suspiciousness score of Block 3 that contains the bug is $\frac{1/1}{3/3+1/1} = 0.5$. Block 1 also has the same suspiciousness score. Interestingly, Block 2 receives the highest suspiciousness score: $\frac{1/1}{2/3+1/1} = 0.6$. Following the same calculation, Blocks 4 and 5 are not suspicious. There is no fail test case that executes these blocks and hence Tarantula returns 0 score. The bug in Block 3 can be localized after inspecting 3 blocks according to the ranks of the scores and the source code order.

2) Ochiai: Abreu et al. [2] suggest Ochiai metric as the suspiciousness score for a program element:

$$suspiciousness(e) = \frac{n_f(e)}{\sqrt{n_f(n_f(e) + n_s(e))}} = \frac{n_f(e)}{\sqrt{n_f n(e)}}$$

Similar to Tarantula, Ochiai considers an element more suspicious if it occurs more frequently in failed executions than in correct executions (the $\sqrt{\frac{n_f(e)}{n(e)}}$ part). Using the same example in Figure 1, Blocks 1 and 3 receive a suspiciousness score: $1/\sqrt{1*(1+3)} = 0.50$. Similar to Tarantula, Ochiai also returns Block 2 as the most suspicious block: $1/\sqrt{1*(1+2)} = 0.58$, while the remaining blocks are not suspicious. Same as Tarantula, Ochiai can help to localize the bug after inspecting through 3 blocks. In our study, we are interested to look for other measures that can localize the bug earlier, and Section IV shows one such example.

B. Dichotomous Association

A common characteristics of the association measures evaluated in this paper is that they are all defined based on dichotomy matrices. The following are the necessary definitions.

*Definition 3.2 (***Dichotomy):** A *dichotomous outcome* is an outcome whose values could be split into two categories, e.g., wrong or correct, executed or skipped, and married or unmarried, etc. A *dichotomous variable* is a variable having a dichotomous outcome. A *dichotomy matrix* is a 2×2 matrix that tries to associate two dichotomous variables in the form of a 2×2 contingency table which records the bivariate frequency distribution of the two variables.

An example of a dichotomy matrix $D(A, B)$ relating variables A and B is shown in Table I. The value c_{00} corresponds to the number of observations in which the value of variable A=A_0 and the value of variable B=B_0. The values in the other three entries in the dichotomy matrix are similar.

	$A = A_0$	$A = A_1$
$B = B_0$	c_{00}	c_{01}
$B = B_1$	c_{10}	c_{11}

TABLE I
AN EXAMPLE OF A DICHOTOMY MATRIX. WE REFER TO IT AS $D(A, B)$.

From a dichotomy matrix, dichotomous associations could be defined.

*Definition 3.3 (***Dichotomous Association):** A *dichotomous association* is a special form of bivariate association [23] which measures the strength of association between two dichotomous variables, e.g., application of a medical treatment and recovery from the disease, job satisfaction and productivity, and program element execution and program failure. The formulae for calculating dichotomous associations depend on the four entries in dichotomy matrices.

Two questions are often asked:

1) Is there a (dichotomous) association between the two variables?
2) How strong is the association between the two variables?

A common way to answer these two questions is to define a formula, called *association measure*, to calculate a score based on the four entries in a dichotomy matrix and consider the association exist or strong if the score is beyond a particular threshold. Then, an accuracy criteria is established to evaluate the quality of the formula. Many well-known formulae based dichotomy matrices have been proposed in the literature since the early 20th century. The following sections describe and evaluate these association measures in more details.

*Definition 3.4 (***Association Measure):** An association measure M is a mathematical function of the four entries of a dichotomy matrix $D(A, B)$, and denoted as $M(A, B, D(A, B))$ or simply $M(A, B, D)$ if causing no confusion.

In our paper, we focus on fault localization and thus are especially interested in the accuracies of these association measures in measuring the strength of the association between the execution or non-execution of a program element and the occurrence of a failure. Thus, we need to consider three random variables for each program element e:

Variable	Definition
E	A program element e is executed
$\overline{E}$	A program element e is not executed
F	A program failure occurs

Also, we are interested in both $M(E, F, D_e(E, F))$ (i.e., the association between the execution of e and a failure) and $M(\overline{E}, F, D_e(\overline{E}, F))$ (i.e., the association between the non-execution of e and a failure), where $D_e(E, F)$ represents the dichotomy matrix of the two variables E and F for e. The next

section presents the construction of the dichotomy matrices and it should be easy to see that $D_e(E, F)$ and $D_e(\overline{E}, F)$ are actually the same since E and $\overline{E}$ are mutually exclusive.

IV. Association Measures

We investigate the effectiveness of 20 association measures in fault localization and compare them against Tarantula and Ochiai. In this section, we first introduce how we establish the dichotomy matrices. We then describe the 20 association measures computed from the dichotomy matrices. Finally, we describe how we transform the association measures to suspiciousness scores for fault localization.

A. Constructing a Dichotomy Matrix

Consider a program element e, which is a basic block in our paper. For each test case, a trace is generated when a subject program is executed on the test case. Some traces go through e (i.e., e is executed), others do not (i.e., e is not executed). Some traces exhibit failures, some produce correct outputs. After the test cases are run, we construct a dichotomy matrix as in Table II for every program element e:

	e Executed	e Not Executed
Test Passed	$n_s(e)$	$n_s(\overline{e})$
Test Failed	$n_f(e)$	$n_f(\overline{e})$

TABLE II
Dichotomy Matrix for Fault Localization.

The notation $\overline{e}$ means e is not executed, and other notations are similar to those mentioned in Section III-A1: $n_s(e)$ corresponds to the number of traces that execute e but do not fail; $n_f(e)$ corresponds to the number of traces that execute e and fail; $n_s(\overline{e})$ is the number of traces that do not execute e and do not fail; $n_f(\overline{e})$ corresponds to the number of traces that do not execute e but fail. Using the notations from Section III-B, $D_e(E, F)$ is obviously the same as $D_e(\overline{E}, F)$ besides the ordering of the columns in the matrices. We simply refer to the two dichotomy matrices associated with the element e as D_e when there is no confusion.

B. Association Measures

The 20 association measures that we consider are: ϕ-coefficient [23], odds ratio [6], Yule's Q [47], Yule's Y [48], Kappa [15], J-Measure [42], gini index [19], support [5], confidence [5], Clark and Boswell's Laplace accuracy [12], conviction [9], interest [9], cosine [45], Piatetsky-Shapiro's Leverage [35], certainty factor [40], added value [45], collective strength [4], Jaccard [21], Klosgen [28], and information gain [10], [37].

The mathematical formulae for calculating these 20 association measures are given in Table III. The formulae are defined in terms of probabilities, instead of frequencies, but we can substitute frequencies recorded in dichotomous matrices for probabilities during actual calculations. The ranges of values that these association measures can take are given in Table IV.

C. From Association to Suspiciousness

Based on the previous subsections, we define the suspiciousness measure for each element e in Definition 4.1.

*Definition 4.1 (***Suspiciousness(e)):** An element e can be a control block (e.g., `if`, `while`, `for` statements) or a non-control block. If e is a non-control block, the *suspiciousness* of e can be defined as the association between the execution of e with failure ($M(E, F, D_e)$). Otherwise, if e is a control block and $children$ is the list of direct children of e in the control flow graph of the containing program, the *suspiciousness* of e is the maximum of the following values:

1 $M(E, F, D_e)$
2 $max_{c \in children}.\ M(\overline{E}, F, D_c)$

The definition uses $M(E, F, D_e)$ to measure the suspiciousness of element e when it is executed. As a matter of experience from previous studies, there are program failures associated with program elements that are not executed, and these errors are often due to faults in the controlling program element (e.g., a wrong condition used in an `if` statement). In this case, the non-executed program elements are the direct children of the controlling program elements. Thus, we calculate the association between the non-execution of the children of a program element with failure ($M(\overline{E}, F, D_c)$). The *suspiciousness* of e is the maximum of the association strength among the children compared to the value of $M(E, F, D_e)$.

*Definition 4.2 (***Example):** Using the example in Figure 1, we observe that Blocks 1, 3, and 4 are the control block of Blocks 2, 4, and 5 respectively, while Blocks 2 and 5 are non-control block. The bug resides at the `if` condition belongs to Block 3. The suspiciousness score of Block 3 is determined by the maximum of the association strength between the execution of Block 3 with the failure and the association strength between the non-execution of Block 4 with the failure. For example, by using one of the association measures, e.g. *cosine*, the suspiciousness score of Block 3 is $max(\frac{0.25}{\sqrt{1 \times 0.25}}, \frac{0.25}{\sqrt{0.5 \times 0.25}}) = 0.71$. Following the similar way, the suspiciousness score of control Blocks 1 and 4 are 0.50 and 0.71 respectively. Since Blocks 2 and 5 are non-control block, the suspiciousness score of Block 2 is calculated as $\frac{0.25}{\sqrt{0.75 \times 0.25}} = 0.50$, and Block 5 is also calculated as $\frac{0}{\sqrt{0.5 \times 0.25}} = 0$ score. Thus, this particular measure with the source code order can help to rank the block that contains the bug receives the highest, while Tarantula and Ochiai do not.

V. Experiments

To further evaluate the wide range of association measures, we analyze different programs from Siemens Test Suite [24]. The test suite was originally used for research in test coverage adequacy and was developed by Siemens Corporation Research. We use the variant provided at www.cc.gatech.edu/aristotle/Tools/subjects/. The test suite contains several programs. Each program contains many different versions where each version has one bug. These bugs comprise a wide array of realistic bugs.

Name	Formula
ϕ-Coefficient (M_1)	$\frac{P(A,B)-P(A)P(B)}{\sqrt{P(A)P(B)(1-P(A))(1-P(B))}}$
Odds ratio (M_2)	$\frac{P(A,B)P(\overline{A},\overline{B})}{P(A,\overline{B})P(\overline{A},B)}$
Yule's Q (M_3)	$\frac{P(A,B)P(\overline{AB})-P(A,\overline{B})P(\overline{A},B)}{P(A,B)P(\overline{AB})+P(A,\overline{B})P(\overline{A},B)} = \frac{\alpha-1}{\alpha+1}$
Yule's Y (M_4)	$\frac{\sqrt{P(A,B)P(\overline{AB})}-\sqrt{P(A,\overline{B})P(\overline{A},B)}}{\sqrt{P(A,B)P(\overline{AB})}+\sqrt{P(A,\overline{B})P(\overline{A},B)}} = \frac{\sqrt{\alpha}-1}{\sqrt{\alpha}+1}$
Kappa(M_5)	$\frac{P(A,B)+P(\overline{A},\overline{B})-P(A)P(B)-P(\overline{A})P(\overline{B})}{1-P(A)P(B)-P(\overline{A})P(\overline{B})}$
J-Measure(M_6)	$\max(P(A,B)\log(\frac{P(B\|A)}{P(B)}) + P(A\overline{B})\log(\frac{P(\overline{B}\|A)}{P(\overline{B})}),$ $P(A,B)\log(\frac{P(A\|B)}{P(A)}) + P(\overline{A}B)\log(\frac{P(\overline{A}\|B)}{P(\overline{A})}))$
Gini Index(M_7)	$\max(P(A)[P(B\|A)^2 + P(\overline{B}\|A)^2] + P(\overline{A})[P(B\|\overline{A})^2 + P(\overline{B}\|\overline{A})^2] - P(B)^2 - P(\overline{B})^2,$ $P(B)[P(A\|B)^2 + P(\overline{A}\|B)^2] + P(\overline{B})[P(A\|\overline{B})^2 + P(\overline{A}\|\overline{B})^2] - P(A)^2 - P(\overline{A})^2)$
Support(M_8)	$P(A,B)$
Confidence(M_9)	$\max(P(B\|A), P(A\|B))$
Laplace(M_{10})	$\max(\frac{P(A,B)+1}{P(A)+2}, \frac{P(A,B)+1}{P(B)+2})$
Conviction(M_{11})	$\max(\frac{P(A)P(\overline{B})}{P(A\overline{B})}, \frac{P(B)P(\overline{A})}{P(B\overline{A})})$
Interest(M_{12})	$\frac{P(A,B)}{P(A)P(B)}$
Cosine(M_{13})	$\frac{P(A,B)}{\sqrt{P(A)P(B)}}$
Piatetsky-Shapiro's(M_{14})	$P(A,B) - P(A)P(B)$
Certainty Factor(M_{15})	$\max(\frac{P(B\|A)-P(B)}{1-P(B)}, \frac{P(A\|B)-P(A)}{1-P(A)})$
Added Value(M_{16})	$\max(P(B\|A) - P(B), P(A\|B) - P(A))$
Collective Strength(M_{17})	$\frac{P(A,B)+P(\overline{AB})}{P(A)P(B)+P(\overline{A})P(\overline{B})} \times \frac{1-P(A)P(B)-P(\overline{A})P(\overline{B})}{1-P(A,B)-P(\overline{AB})}$
Jaccard(M_{18})	$\frac{P(A,B)}{P(A)+P(B)-P(A,B)}$
Klosgen(M_{19})	$\sqrt{P(A,B)}\max(P(B\|A) - P(B), P(A\|B) - P(A))$
Information Gain (M_{20})	$(-P(B)\log P(B) - P(\overline{B})\log P(\overline{B}))-$ $(P(A) \times (-P(B\|A)\log P(B\|A)) - P(\overline{B}\|A)\log P(\overline{B}\|A)-$ $P(\overline{A}) \times (-P(B\|\overline{A})\log P(B\|\overline{A})) - P(\overline{B}\|\overline{A})\log P(\overline{B}\|\overline{A})))$

TABLE III

DEFINITIONS OF ASSOCIATION MEASURES. A AND B ARE THE TWO VARIABLES IN THE DICHOTOMY MATRIX. $P(A)$ AND $P(B)$ CORRESPOND TO THE PROBABILITIES OF A AND B RESPECTIVELY. OTHER NOTATIONS FOLLOW STANDARD NOTATIONS IN PROBABILITY AND STATISTICS: $P(\overline{A})$ IS THE PROBABILITY OF *not* A; $P(A,B)$ IS THE JOINT PROBABILITY OF A AND B; $P(A|B)$ AND $P(B|A)$ ARE CONDITIONAL PROBABILITIES.

Name	Range	No	Perfect
ϕ-Coefficient (M_1)	$-1\ldots0\ldots1$	0.0	1.0
Odds ratio (M_2)	$0\ldots1\ldots\infty$	1.0	∞
Yule's Q (M_3)	$-1\ldots0\ldots1$	0.0	1.0
Yule's Y (M_4)	$-1\ldots0\ldots1$	0.0	1.0
Kappa (M_5)	$-1\ldots0\ldots1$	0.0	1.0
J_Measure (M_6)	$0\ldots1$	0.0	1.0
Gini index (M_7)	$0\ldots1$	0.0	1.0
Support (M_8)	$0\ldots1$	0.0	1.0
Confidence (M_9)	$0\ldots1$	0.0	1.0
Laplace (M_{10})	$0\ldots1$	0.0	1.0
Conviction (M_{11})	$0.5\ldots1\ldots\infty$	1	∞
Interest (M_{12})	$0\ldots1\ldots\infty$	1.0	∞
Cosine (M_{13})	$0\ldots\sqrt{P(A,B)}\ldots1$	$\sqrt{P(A,B)}$	1.0
Piatetsky-Shapiro's (M_{14})	$-0.25\ldots0\ldots0.25$	0	0.25
Certainty factor (M_{15})	$-1\ldots0\ldots1$	0.0	1.0
Added Value (M_{16})	$-0.5\ldots0\ldots1$	0.0	1.0
Collective strength (M_{17})	$0\ldots1\ldots\infty$	1	∞
Jaccard (M_{18})	$0\ldots1$	0.0	1.0
Klosgen (M_{19})	$(\frac{2}{\sqrt{3}}-1)^{1/2}[2-\sqrt{3}-\frac{1}{\sqrt{3}}]\ldots0\ldots\frac{2}{3\sqrt{3}}$	0	$\frac{2}{3\sqrt{3}}$
Information Gain (M_{20})	$0\ldots1$	0.0	1.0

TABLE IV

ASSOCIATION MEASURE RANGES. THE THIRD AND FOURTH COLUMNS GIVE THE VALUE CORRESPONDING TO NO ASSOCIATION AND PERFECT ASSOCIATION RESPECTIVELY. MEASURES THAT HAVE TWO VALUES AS THEIR RANGE (E.G., J-MEASURE,ETC.), INDICATE POSITIVE ASSOCIATION WITH FAILURES. A LARGER VALUE WITHIN THE RANGE INDICATES A STRONGER ASSOCIATION WITH FAILURES. MEASURES THAT HAVE THREE VALUES AS THEIR RANGE (E.G., ϕ-COEFFICIENT (-1..0..1)), INDICATES POSITIVE ASSOCIATION WITH FAILURES FOR VALUES BETWEEN 0 TO 1, AND NEGATIVE ASSOCIATION FOR VALUES BETWEEN -1 TO 0. SMALLER NEGATIVE VALUES IMPLY A STRONGER ASSOCIATED WITH SUCCESSFUL EXECUTIONS.

Dataset	LOC	No. of Faulty Version	No. of Test cases
print_token	478	7	4130
print_token2	399	10	4115
replace	512	32	5542
schedule	292	9	2650
schedule2	301	10	2710
tcas	141	41	1608
tot_info	440	23	1052

TABLE V
EXPERIMENT DATASET

Association Measures	Mean	StdDev
ϕ-Coefficient (M_1)	0.31	0.30
Odds ratio (M_2)	0.55	0.18
Yule's Q (M_3)	0.54	0.18
Yule's Y (M_4)	0.54	0.18
Kappa (M_5)	0.34	0.29
J-Measure (M_6)	0.47	0.26
Gini Index (M_7)	0.62	0.29
Support (M_8)	0.55	0.17
Confidence (M_9)	0.34	0.29
Laplace (M_{10})	0.55	0.17
Conviction (M_{11})	0.54	0.18
Interest (M_{12})	0.34	0.29
Cosine (M_{13})	0.29	0.28
Piatetsky-Shapiro's (M_{14})	0.84	0.27
Certainty Factor (M_{15})	0.52	0.18
Added Value (M_{16})	0.31	0.30
Collective Strength (M_{17})	0.31	0.29
Jaccard (M_{18})	0.32	0.29
Klosgen (M_{19})	0.30	0.29
Information Gain (M_{20})	0.28	0.29
Tarantula	0.32	0.29
Ochiai	0.28	0.28

TABLE VI
OVERALL MEAN AND STANDARD DEVIATION (IN PARENTHESES) OF ACCURACY VALUES (SMALLER THE BETTER)

The Siemens Test Suite comes with 7 programs: print_tokens, print_tokens2, replace, schedule, schedule2, tcas, and tot_info. The total number of buggy versions are 132, as shown in Table V. We manually instrumented the buggy versions using basic block level. Since our instrumentation cannot reach the bug that resides in variable declaration, we exclude versions that contain this type of bug e.g. version 6, 10, 19, 21 of tot_info dataset, version 12 of replace dataset, and version 13, 14, 15, 36, 38 of tcas dataset. We also exclude version 4 and 6 of print_token because they are identical with the original version. Thus, we use 120 buggy versions in total.

A. Evaluation Metric

We evaluate the performance of the measures by the number of elements that are ranked as high or higher than the program element containing the fault/the bug. For each version, the suspiciousness score of all program elements are sorted in descending order. For a suspiciousness score to be effective, buggy program elements should have a relatively larger value of suspiciousness scores than the non-buggy elements. Based on the descending order of suspiciousness scores, we then rank the various program elements.

When a buggy program element has the same suspiciousness score with several other elements, the largest rank of the elements that has this suspiciousness score is used as the rank of the buggy element. For example, consider the case where the two highest suspiciousness scores are 0.92 and 0.91 where each score has two and three elements that have the same score respectively. If a buggy element is given the suspiciousness score of 0.91, then the rank of this buggy element is 5, instead of 3. Since we do not know how the programmer will traverse elements that have the same suspiciousness score, we use the worst case scenario where the programmer inspects the buggy element at the last position. In the situation when a version contains a bug that covers several program elements, we use the largest rank among the buggy elements as the rank of this version.

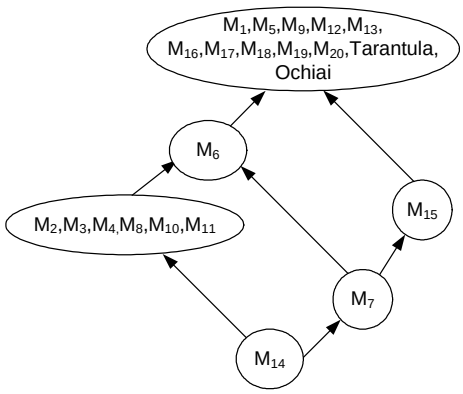

Fig. 2. Accuracy Partial Order

Suspiciousness measures that rank the buggy elements first are more effective than those that rank them last. We then use this rank to compute the percentage of the program elements that need to be inspected to find the bug by the formula:

$$\frac{rank}{total\ elements}$$

In our experiment, we use basic block as the granularity of the element and the above percentage is applied as the accuracy criterion of the association measures.

B. Experiment Results

The overall mean and standard deviation of the accuracy values of the 20 association measures along with those of Ochiai and Tarantula for the 7 programs in Siemens Test Suite are shown in Table VI. The smallest mean of accuracy value (0.28) is achieved by Ochiai and information gain, while Tarantula achieved 0.32. Some measures that have similar accuracy as the above are cosine(0.29), Klosgen(0.30) and three measures with accuracy equals to 0.31 (i.e., collective strength, added value, ϕ-coefficient). The detail of the accuracy values for each dataset are shown in Table VII.

We also plot the curve showing the size of the code that needs to be traversed to find the bug (x-axis) vs. the proportion of bugs localized (y-axis). We split the large graphs into several smaller graphs as shown in Figures 3, 4, 5, and 6. For each graph, we compare several association measures with Tarantula and Ochiai. Association measures included in Figure 3 and Figure 4 do not perform better as compare to Tarantula and Ochiai. Some of the association measures (e.g. odds ratio(M_2),

		Programs						
		print_token	print_token2	replace	schedule	schedule2	tcas	tot_info
Association Measures	ϕ-Coefficient (M_1)	0.18 (0.23)	0.12 (0.17)	0.14 (0.21)	0.23 (0.32)	0.59 (0.32)	0.51 (0.28)	0.24 (0.16)
	Odds ratio (M_2)	0.41 (0.08)	0.44 (0.03)	0.42 (0.15)	0.64 (0.23)	0.66 (0.20)	0.66 (0.13)	0.53 (0.17)
	Yule's Q (M_3)	0.41 (0.08)	0.43 (0.03)	0.42 (0.15)	0.64 (0.23)	0.66 (0.20)	0.65 (0.13)	0.53 (0.17)
	Yule's Y (M_4)	0.42 (0.07)	0.43 (0.03)	0.42 (0.15)	0.64 (0.23)	0.66 (0.20)	0.65 (0.13)	0.53 (0.17)
	Kappa (M_5)	0.27 (0.25)	0.14 (0.20)	0.17 (0.22)	0.23 (0.30)	0.62 (0.32)	0.52 (0.28)	0.29 (0.18)
	J-Measure (M_6)	0.28 (0.25)	0.45 (0.08)	0.43 (0.26)	0.77 (0.21)	0.40 (0.30)	0.45 (0.24)	0.50 (0.28)
	Gini Index (M_7)	0.47 (0.27)	0.67 (0.18)	0.63 (0.27)	0.93 (0.10)	0.48 (0.30)	0.61 (0.29)	0.54 (0.31)
	Support (M_8)	0.44 (0.08)	0.45 (0.03)	0.43 (0.14)	0.62 (0.21)	0.60 (0.18)	0.65 (0.13)	0.54 (0.17)
	Confidence (M_9)	0.28 (0.25)	0.10 (0.15)	0.18 (0.22)	0.22 (0.26)	0.55 (0.31)	0.53 (0.28)	0.31 (0.18)
	Laplace (M_{10})	0.44 (0.08)	0.45 (0.03)	0.45 (0.16)	0.62 (0.21)	0.60 (0.18)	0.65 (0.13)	0.54 (0.18)
	Conviction (M_{11})	0.41 (0.08)	0.43 (0.03)	0.42 (0.15)	0.63 (0.23)	0.65 (0.20)	0.65 (0.13)	0.53 (0.17)
	Interest (M_{12})	0.28 (0.25)	0.15 (0.19)	0.18 (0.22)	0.23 (0.26)	0.55 (0.31)	0.52 (0.28)	0.30 (0.18)
	Cosine (M_{13})	0.14 (0.17)	0.09 (0.11)	0.14 (0.20)	0.26 (0.36)	0.49 (0.29)	0.49 (0.26)	0.21 (0.14)
	Piatetsky-Shapiro's (M_{14})	0.56 (0.46)	0.89 (0.11)	0.89 (0.24)	0.94 (0.08)	0.84 (0.28)	0.82 (0.32)	0.78 (0.22)
	Certainty Factor (M_{15})	0.38 (0.10)	0.43 (0.03)	0.39 (0.15)	0.60 (0.25)	0.65 (0.21)	0.62 (0.13)	0.52 (0.17)
	Added Value (M_{16})	0.17 (0.23)	0.12 (0.18)	0.14 (0.21)	0.21 (0.31)	0.62 (0.33)	0.52 (0.29)	0.23 (0.17)
	Collective Strength (M_{17})	0.19 (0.21)	0.13 (0.17)	0.15 (0.21)	0.25 (0.34)	0.56 (0.32)	0.51 (0.27)	0.23 (0.16)
	Jaccard (M_{18})	0.26 (0.25)	0.09 (0.15)	0.17 (0.22)	0.22 (0.26)	0.55 (0.31)	0.52 (0.28)	0.27 (0.16)
	Klosgen (M_{19})	0.22 (0.27)	0.09 (0.13)	0.13 (0.21)	0.21 (0.32)	0.62 (0.29)	0.50 (0.28)	0.22 (0.15)
	Information Gain (M_{20})	0.04 (0.04)	0.10 (0.11)	0.16 (0.24)	0.23 (0.31)	0.47 (0.37)	0.47 (0.28)	0.19 (0.12)
	Tarantula	0.30 (0.27)	0.15 (0.18)	0.16 (0.22)	0.17 (0.28)	0.62 (0.30)	0.51 (0.27)	0.22 (0.16)
	Ochiai	0.14 (0.16)	0.09 (0.11)	0.12 (0.20)	0.23 (0.37)	0.55 (0.32)	0.48 (0.24)	0.17 (0.12)

TABLE VII
DETAILED MEAN AND STANDARD DEVIATION (IN PARENTHESES) OF ACCURACY VALUES

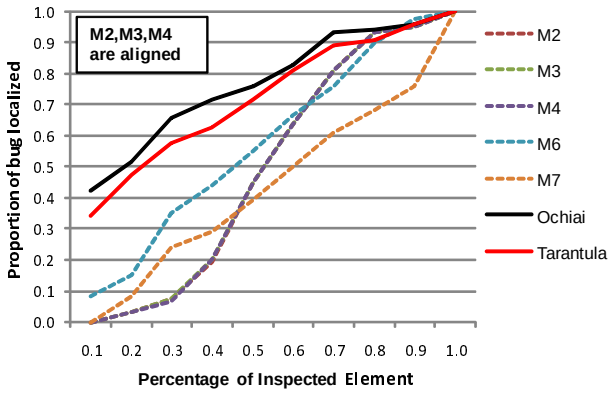

Fig. 3. Comparing M_2, M_3, M_4, M_6, M_7 With Ochiai and Tarantula

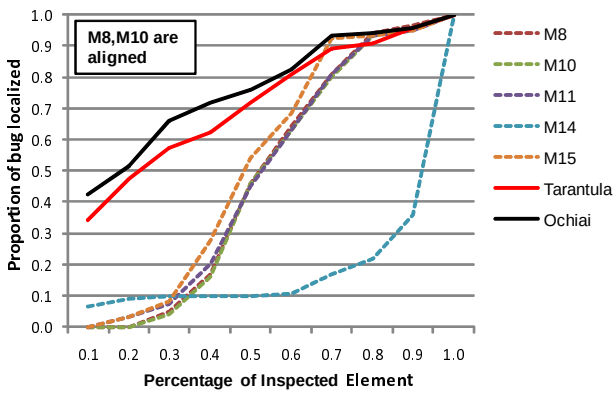

Fig. 4. Comparing M_8, M_{10}, M_{11}, M_{14}, M_{15} With Ochiai and Tarantula

Yule's $Q(M_3)$, Yule's $Y(M_3)$, J-Measure(M_6), support(M_8), laplace(M_{10}), conviction(M_{11}), certainty factor (M_{15})) can localize more bugs compare to Tarantula when after 70% program elements have been inspected.

In Figures 5 and 6, most of the association measures perform comparably to Tarantula and Ochiai. Based on the proportion of bugs localized, Kappa(M_5), confidence(M_9), and interest(M_{12}) perform slightly worse than Tarantula. On the other hand, ϕ-coefficient(M_1), added value(M_{16}), collective strength(M_{17}), Jaccard(M_{18}), and Klosgen(M_{19}) perform slightly better than Tarantula even though not as good as Ochiai. We also notice that added value(M_{16}) has a similar performance with Ochiai when 10% of the program elements are inspected. At 10%, the proportion of bugs localized by added value (M_{16}) is 42%, while Ochiai achieves 43%. Other measure that performs almost similarly to Ochiai is cosine(M_{13}). Information gain(M_{20}) performs slightly worse than Ochiai when less than 10% of the program elements are inspected. At 10%, Information gain (M_{20}) localizes 37% of the bugs. However, at 20%, it localizes 4% more bugs than Ochiai. At 50%, 83% of the bugs are localized by information gain, which is 8% higher than that of Ochiai. Thus, based on this result, a number of association measures can perform better than Tarantula. Also, information gain(M_{20}) outperforms both Tarantula and Ochiai when more than 10% program elements are inspected.

We also perform statistical tests for each pair of measures including Tarantula and Ochiai (i.e., two sample unpooled t-test at 0.05 statistical significance threshold) to see if some measures are statistically significantly better than others. We plot this as a partial order in Figure 2. It is interesting to note Tarantula and Ochiai are comparable (no one statistically significantly outperforms the other). It is also interesting to note that 10 measures perform comparably as Tarantula and Ochiai. These are: ϕ-coefficient (M_1), Kappa (M_5), confidence (M_9), interest (M_{12}), cosine (M_{13}), added value (M_{16}), collective strength (M_{17})), Jaccard (M_{18}), Klosgen (M_{19})), and information gain (M_{20}). Based on partial order, certainty factor(M_{15}) is not comparable with odds ratio(M_2), neither with J-Measure (M_6). Also it could be noted that Piatetsky-

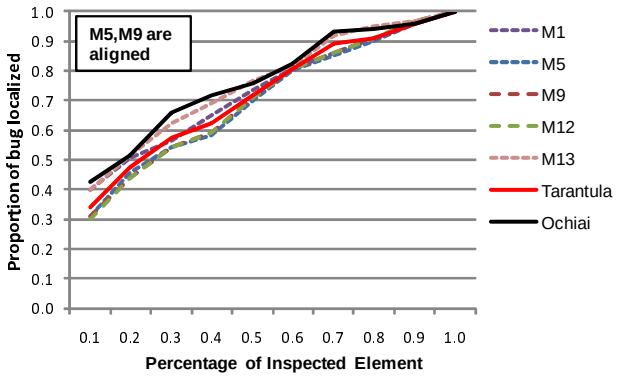

Fig. 5. Comparing M_1, M_5, M_9, M_{12}, M_{13} With Ochiai and Tarantula

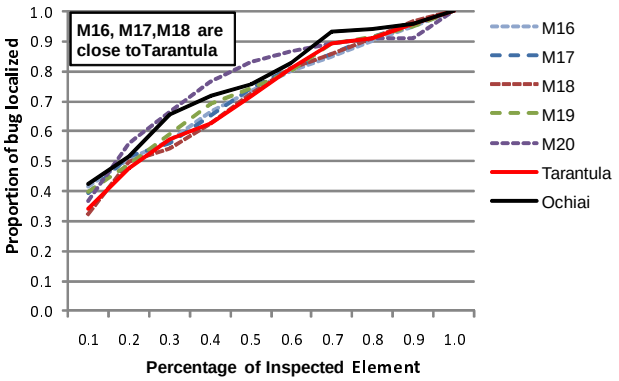

Fig. 6. Comparing M_{16}-M_{20} With Ochiai and Tarantula

Shapiro's (M_{14}) performs worse than other association measures for bug localization.

C. Discussion

In this section, we discuss the answer to the research questions mentioned in Section I.

RQ1. We are interested to find if off-the-shelf association measures are powerful enough to locate bugs. Based on the mean accuracy values of the measures, it could be noted that the various association measures could find the bugs when 28 - 84% of the program elements have been inspected. Fifty percent of the association measures are able to find bugs by inspecting 28-35% of elements, while Tarantula and Ochiai require to inspect approximately 28% and 32% of the elements respectively.

RQ2. Next, we are interested to find which association measures are better than others. The answer to this research question is the partial order shown in Figure 2. At the top of the partial order there are 10 off-the-shelf association measures namely: ϕ-coefficient, Kappa, confidence, interest, cosine, added value, collective strength, Jaccard, Klosgen, and information gain. They are statistically significantly better than the other off-the-shelf association measures.

RQ3. Finally, we would like to know the relative improvement of the association measures versus well-known suspiciousness measures for bug localization. By applying statistical significance tests under 0.05 significance threshold, the top 10 association measures are comparable to Tarantula and Ochiai. They are not statistically significantly worse than the two measures. Based on the proportion of bugs localized, information gain localizes 4% more bugs as compared to Ochiai when 20% of the program elements are inspected; when 50% of the program elements are inspected, 83% of the bugs are localized which is 8% more than those localized by Ochiai.

D. Threats to Validity

The accuracy of a measure to localize a bug based on spectrum fault localization is influenced by the granularity level of the instrumented program (e.g. statement, basic block, or method level). Different granularity levels may produce different accuracies since they would have different total number of elements that would affect the percentage value of inspected elements. In our paper, we only instrument basic blocks. The instrumented versions that we use in this work are manually instrumented. To maintain the consistency, only one person manually instruments the versions. Although another check was performed, human error is still possible. In addition, we only analyze Siemens programs which are a collection of small C programs. This poses a threat to external validity especially on the generalizability of our approach to larger programs written in various programming languages.

VI. Conclusion & Future Work

In this work, we investigate a comprehensive number of association measures proposed in the literature. These measures gauge the strength of association between two variables expressible as a dichotomy matrix. We consider and compare 20 association measures. The well-known fault-localization measures namely Tarantula and Ochiai are used as baselines. We have conducted a number of statistical significant tests. Interestingly, Tarantula and Ochiai are comparable to each other; one does not statistically significantly outperforms the other. We also notice that 10 of the association measures are comparable (not statistically significantly under performs Tarantula and Ochiai). The comparable association measures with Tarantula and Ochiai are cosine, ϕ-coefficient, added value, collective strength, Klosgen, Kappa, Jaccard, confidence, interest, and information gain. Thus, we can conclude that association measures are also promising to be used in fault localization problem.

In the future, we plan to investigate the association measures on larger programs and characterize the effectiveness of each measures on different types of bugs.

Dataset, Tool & Acknowledgement. Our dataset and tool are made publicly available at: http://www.mysmu.edu/phdis2009/lucia.2009/Dataset.htm. We would like to thank the anonymous reviewers for their inputs and advice.

References

[1] R. Abreu, "Spectrum-based fault localization in embedded software." Ph.D. dissertation, Delft University of Technology, 2009.

[2] R. Abreu, P. Zoeteweij, and A. J. C. van Gemund, "On the Accuracy of Spectrum-based Fault Localization," in *Mutation Testing: Academic and Industrial Conference Practice and Research Techniques (TAICPART-MUTATION)*, 2007.

[3] R. Abreu, P. Zoeteweij, and A. J. van Gemund, "Spectrum-Based Multiple Fault Localization," in *IEEE/ACM International Conference on Automated Software Engineering*, Auckland, New Zealand, 2009.

[4] C. Aggarwal and P. S. Yu, "A new framework for itemset generation." in *Symposium on Principles of Database Systems (PODS)*, 1998.

[5] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules," in *Proc. of Int. Conf. on Very Large Data Bases*, 1994.

[6] A. Agresti, *An Introduction to Categorical Data Analysis*. John Wiley & Sons, 1996.

[7] R. B. Altman and T. E. Klein, "Challenges for biomedical informatics and pharmacogenomics," *Annu. Rev. Pharmacol. Toxicol.*, vol. 42, pp. 113–133, 2002.

[8] B. Beizer, *Software Testing Techniques*, 2nd ed. Boston: International Thomson Computer Press, 1990.

[9] S. Brin, R. Motwani, J. D. Ullman, and S. Tsur, "Dynamic itemset counting and implication rules for market basket analysis," in *SIGMOD*, 1997, pp. 255–264.

[10] H. Cheng, D. Lo, Y. Zhou, X. Wang, and X. Yan, "Identifying bug signatures using discriminative graph mining," in *ISSTA*, 2009.

[11] T. M. Chilimbi, B. Liblit, K. Mehra, A. V. Nori, and K. Vaswani, "Holmes: Effective statistical debugging via efficient path profiling," in *ICSE*, 2009, pp. 34–44.

[12] P. Clark and R. Boswell, "Rule induction with cn2: Some recent improvements." in *In Machine Learning - EWSL-91*, 1991, p. 151163.

[13] H. Cleve and A. Zeller, "Locating causes of program failures," in *ICSE*, 2005, pp. 342–351.

[14] C.Liu, X. Yan, L. Fei, J. Han, and S. Midkiff, "Sober: Statistical model-based bug localization," in *ESEC/FSE*, Lisbon, Portugal, Sep. 2005.

[15] J. Cohen, "A coefficient of agreement for nominal scales," *Educational and Psychological Measurement*, vol. 20, no. 1, pp. 37–46, 1960.

[16] H. Do, S. G. Elbaum, and G. Rothermel, "Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact." *Empirical Software Engineering: An International Journal*, vol. 10, no. 4, pp. 405–435, 2005.

[17] A. Feldman and A. van Gemund, "A two-step hierarchical algorithm for model-based diagnosis," in *Proceedings of the 21st National Conference on On Artificial Intelligence*. Boston, Massachusetts: AAAI Press, 2006, pp. 827–833.

[18] L. Geng and H. Hamilton, "Interestingness measures for data mining: A survey." in *ACM Computing Surveys*, 2006.

[19] C. Gini, "Variability and mutability, contribution to the study of statistical distributions and relations." *Studi Economico-Giuricici della R. Universita de Cagliari*, vol. 3(part 2), no. i-iii, pp. 3–159, 1912.

[20] N. Gupta, H. He, X. Zhang, and R. Gupta, "Locating faulty code using failure-inducing chops," in *ASE*, 2005, pp. 263–272.

[21] D. J. Hand, H. Mannila, and P. Smyth, *Principles of Data Mining*. MIT Press, 2001.

[22] M. Harrold, G. Rothermel, K. Sayre, R. Wu, and L. Yi, "An empirical investigation of the relationship between spectra differences and regression faults." *Software Testing, Verification and Reliability*, vol. 10, no. 3, pp. 171–194, 2000.

[23] J. F. Healey, *Statistics: A Tool for Social Research*, 8th ed. Wadsworth Publishing, 2008. [Online]. Available: http://www.amazon.com/Statistics-Research-Joseph-F-Healey/dp/0495096555

[24] M. Hutchins, H. Foster, T. Goradia, and T. Ostrand, "Experiments of the effectiveness of dataflow- and controlflow-based test adequacy criteria," in *Proc. of ICSE*, 1994, pp. 191–200.

[25] D. Jeffrey, N. Gupta, and R. Gupta, "Fault localization using value replacement," in *International Symposium on Software Testing and Analysis*, 2008.

[26] J. Jones, M. Harrold, and J. Stasko, "Visualization of test information to assist fault detection," in *Proc. of Internation Conference on Software Engineering*, Orlando, Florida, May. 2002, pp. 467–477.

[27] J. Jones and M. Harrold, "Empirical evaluation of the tarantula automatic fault-localization technique," in *Proc. of International Conference on Automated Software Engineering*, 2005.

[28] W. Klosgen, "Explora: A multipattern and multistrategy discovery assistant." in *Advances in Knowledge Discovery and Data Mining*, 1996.

[29] A. J. Ko and B. A. Myers, "Debugging reinvented:asking and answering why and why not questions about program behavior," in *International Conference on Software Engineering*, 2008.

[30] B. Liblit, A. Aiken, A. X. Zheng, and M. I. Jordan, "Bug isolation via remote program sampling," in *Proc. ACM SIGPLAN 2003 Conf. Programming Language Design and Implementation (PLDI 2003)*, San Diego, CA, June 2003, pp. 141–154.

[31] B. Liblit, M. Naik, A. X. Zheng, A. Aiken, and M. I. Jordan, "Scalable statistical bug isolation," in *Proc. ACM SIGPLAN 2005 Int. Conf. Programming Language Design and Implementation (PLDI'05)*, June 2005.

[32] B. Liblit and A. Aiken, "Building a better backtrace: Techniques for postmortem program analysis," UC Berkeley, Tech. Rep. CSD-02-1203, 2002.

[33] R. Manevich, M. Sridharan, S. Adams, M. Das, and Z. Yang, "PSE: Explaining program failures via postmortem static analysis," in *FSE*, 2004. [Online]. Available: citeseer.ist.psu.edu/manevich04pse.html

[34] W. Mayer and M. Stumptner, "Model-Based Debugging – State of the Art And Future Challenges," *Electronic Notes in Theoretical Computer Science (ENTCS)*, vol. 174, no. 4, 2007.

[35] G. Piatetsky-Shapiro, "Discovery, analysis, and presentation of strong rules." in *KDD*, 1991.

[36] D. Qi, A. Roychoudhury, Z. Liang, and K. Vaswani, "Darwin: An approach for debugging evolving programs," in *ESEC / SIGSOFT FSE '09*. Amsterdam, The Netherlands: ACM, 2009.

[37] J. Quinlan, *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

[38] M. Renieris and S. Reiss, "Fault localization with nearest neighbor queries," in *Proc. of Int. Conf. on Automated Software Engineering*, 2003, pp. 141–154.

[39] T. Reps, T. Ball, M. Das, and J. Larus, "The use of program profiling for software maintenance with applications to the year 2000 problem." in *ESEC/FSE*, 1997.

[40] E. Shortliffe and B. Buchanan, "A model of inexact reasoning in medicine." *Mathematical Biosciences*, vol. 23, p. 351379, 1975.

[41] Siemens, M. J. Harrold, and G. Rothermel, *Aristotle Analysis System – Siemens Programs, HR Variants*, http://www.cc.gatech.edu/aristotle/Tools/subjects/.

[42] P. Smyth and R. Goodman, "An information theoretic approach to rule induction from databases," *IEEE Trans. Knowledge and Data Eng.*, vol. 4, no. 4, pp. 301–316, 1992.

[43] C. D. Sterling and R. A. Olsson, "Automated bug isolation via program chipping," *Software: Practice and Experience (SP&E)*, vol. 37, no. 10, pp. 1061–1086, August 2007, john Wiley & Sons, Inc.

[44] S. Tallam, C. Tian, and R. Gupta, "Dynamic slicing of multithreaded programs for race detection." in *Proc. of ICSM*, 2008.

[45] P.-N. Tan, V. Kumar, and J. Srivastava, "Selecting the right interestingness measure for association patterns," in *KDD*, 2002, pp. 32–41.

[46] G. Tassey, "The economic impacts of inadequate infrastructure for software testing." *National Institute of Standards and Technology. Planning Report 02-3.2002*, 2002.

[47] G. U. Yule, "On the association of attributes in statistics," *Philosophical Transcriptions of the Royal Society*, vol. A194, pp. 257–319, 1900.

[48] ——, "On the methods of measuring association between two attributes," *Journal of the Royal Statistical Society*, vol. 75, pp. 579–642, 1912.

[49] A. Zeller and R. Hildebrandt, "Simplifying and isolating failure-inducing input," *IEEE Transaction on Software Engineering*, vol. 28, pp. 183–200, 2002.

[50] A. Zeller, "Isolating cause-effect chains from computer programs," in *FSE*, 2002, pp. 1–10.

[51] ——, *Why Programs Fail: A Guide to Systematic Debugging*, 2nd ed., T. Cox, Ed. Morgan Kaufmann, 2009.

[52] X. Zhang, N. Gupta, and R. Gupta, "Locating faults through automated predicate switching," in *ICSE*, 2006.