An Empirical Study on the Usage of Testability Information to Fault Localization in Software [*]

Alberto Gonzalez-Sanchez[†] Rui Abreu[‡] Hans-Gerhard Gross[†] Arjan J.C. van Gemund[†]

[†]Department of Software Technology
Delft University of Technology
The Netherlands
{a.gonzalezsanchez, h.g.gross, a.j.c.vangemund}@tudelft.nl

[‡]Department of Informatics Engineering
Faculty of Engineering, University of Porto
Portugal
rui@computer.org

ABSTRACT

When failures occur during software testing, automated software fault localization helps to diagnose their root causes and identify the defective statements of a program to support debugging. Diagnosis is carried out by selecting test cases in such way that their pass or fail information will narrow down the set of fault candidates, and, eventually, pinpoint the root cause. An essential ingredient of effective and efficient fault localization is knowledge about the false negative rate of tests, which is related to the rate at which defective statements of a program will exhibit failures. In current fault localization processes, false negative rates are either ignored completely, or merely estimated *a posteriori* as part of the diagnosis. In this paper, we study the reduction in diagnosis effort when false negative rates are known *a priori*. We deduce this information from testability, following the propagation-infection-execution (PIE) approach. Experiments with real programs suggest significant improvement in the diagnosis process, both in the single and the multiple-fault cases. When compared to the next-best technique, PIE-based false negative rate information yields a fault localization effort reduction of up to 80% for systems with only one fault, and up to 60% for systems with multiple faults.

1. INTRODUCTION

Testing is the most commonly used method for detecting the presence of faults in software. However, once the presence of a fault has been detected (by means of a failing test), its precise location has to be determined. Fault localization denotes the process of finding the root-cause of failures through diagnosis to support debugging. Diagnostic accuracy is a critical success factor in the cycle of testing, diagnosing, acting/recovering, and repairing. Typically multiple diagnoses are possible, and further tests are executed to narrow down the set of possibilities, ideally until a perfectly accurate diagnosis is reached.

In software, not all test inputs will produce a failure, because of the different paths taken in a test, introducing *false negatives* or *coincidental correctness*. Ignoring false negatives, as done in statistical diagnosis approaches [2, 11], results in degraded diagnostic performance [3]. An alternative approach is to approximate the false negative rates (FNR) during a Bayesian diagnosis process [4, 6]. However, the absence of prior knowledge about the false negative rates of faulty statements leads to a loss of diagnostic accuracy. The diagnosis algorithm must approximate the FNR while it computes the diagnosis, and that, again, depends on the FNR. The net result is that a large number of tests is required to obtain an accurate estimate of the FNR, thereby reducing the rate at which the diagnosis converges, and thus, increasing both testing effort and diagnostic effort.

In software fault localization, the FNR depends on the testability of the faulty statements, and can be determined *a priori*, using testability quantification techniques [17, 18]. By providing testability-based information on the false negative rate, the estimation problem can be detached from the diagnosis, leading to significant performance gains in the fault localization process.

In this paper we evaluate to what extent testability information can lead to performance gains in the fault localization process, both in single-fault and multiple-fault settings. We study and assess the testability of the statements in a system, and use that information as input to a Bayesian diagnosis algorithm. As existing techniques for testability quantification are performed in a single-fault setting. Multiple-fault diagnosis relies on failures to be independent events, a condition that may not hold in software. The extent to which testability results can be used for diagnosing systems where multiple faults are present simultaneously remained unknown, and is also evaluated in this paper.

In particular, the paper makes the following contributions.

1. We compare the performance of Bayesian diagnosis with prior FNR information with (1) Bayesian diagnosis without prior knowledge of the FNR, and (2) with similarity coefficients. We perform our experiments in both the single and multiple-fault cases.

2. We evaluate to which extent testability estimations can be used to accurately predict the rate of false negatives when testing a system with multiple faults, i.e.,

[*]This work has been carried out as part of the Poseidon project under the responsibility of the Embedded Systems Institute (ESI), Eindhoven, The Netherlands. This project is partially supported by the Dutch Ministry of Economic Affairs under the BSIK03021 program.

	Program: Character Counter	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8
c_0		0	0	0	0	0	0	0	0
c_1	`main() {`	1	1	1	1	1	1	1	1
c_2	`int let, dig, other, c;`	1	1	1	1	1	1	1	1
c_3	`let = dig = other = 0;`	1	1	1	1	1	1	1	1
c_4	`while(c = getchar()) {`	1	1	1	1	1	1	1	1
c_5	`if ('A'<=c && 'Z'>=c)`	1	1	1	1	1	1	1	0
c_6	`let += 2; /* FAULT */`	1	0	1	1	0	0	1	0
c_7	`elif ('a'<=c && 'z'>=c)`	1	1	0	1	1	1	1	0
c_8	`let += 1;`	1	0	0	0	1	0	1	0
c_9	`elif ('0'<=c && '9'>=c)`	1	1	0	1	1	1	0	0
c_{10}	`dig += 1;`	1	1	1	1	0	0	0	0
c_{11}	`elif (isprint(c))`	0	0	0	0	1	1	0	0
c_{12}	`other += 1;}`	0	0	0	0	1	0	0	0
c_{13}	`printf("%d %d %d\n", let, dig, others);}`	1	1	1	1	1	1	1	1
	Test case outcomes	1	0	1	1	0	0	1	0

Table 1: Faulty program and fault diagnosis inputs

whether failures of different faults are independent.

Our results show that the gains achieved by introducing prior knowledge about false negative rates are significant. For real programs containing real faults, we observed up to a 80% diagnostic effort reduction for the same testing effort in the single fault case, and a 62% for multiple faults.

The paper is organized as follows. Section 2 introduces the main concepts of diagnosis and the main diagnostic techniques. Section 3 presents two different solutions for estimating the FNR of a system under test. The systems used in our experiments are presented in Section 4. An empirical validation is performed in Section 5 for single faults and in Section 6 for multiple faults. Section 7 discusses the threats to the validity of our results. Section 8 describes related work in the area of fault diagnosis. Section 9 concludes the paper and presents our future research directions.

2. FAULT DIAGNOSIS

Consider the example system presented in Table 1, which shows an example faulty program [10]. The objective of fault diagnosis is to pinpoint the precise location of the fault (or faults) in the program by observing the program's behavior given a number of tests.

The following inputs are usually involved in automated diagnosis:

- A finite set $\mathcal{C} = \{c_1, c_2, \ldots, c_j, \ldots, c_M\}$ of M components (e.g., source code statements, function points, classes, etc.) which are potentially faulty. We will denote the number of faults in the system as M_f.
- A finite set $\mathcal{T} = \{t_1, t_2, \ldots, t_i, \ldots, t_N\}$ of N tests with binary outcomes $O = (o_1, o_2, \ldots, o_i, \ldots, o_N)$, where $o_i = 1$ if test t_i failed, and $o_i = 0$ otherwise.
- A $N \times M$ coverage matrix, $A = [a_{ij}]$, where $a_{ij} = 1$ if test t_i involves component c_j, and 0 otherwise.

For the remainder of this paper, we will consider statement level as our level of granularity. Hence, the output of fault localization, is a *diagnosis*, i.e., a list or a statement ranking of statement indices ordered by the likelihoods of each of them being faulty. Diagnostic performance is accounted in terms of a metric C_d that measures the percentage of excess *effort* incurred in finding all statements at fault [4]. C_d measures *wasted effort*, independent of the number of faults M_f in the program, to enable an unbiased evaluation of the effect of M_f on C_d. Thus, regardless of M_f, $C_d = 0$ represents an ideal diagnosis technique (all M_f faulty statements on top of the ranking, no effort wasted on testing other statements to find they are not faulty), while $C_d = M - M_f$ represents the worst case (testing all $M - M_f$ healthy statements until arriving at the M_f faulty ones).

For example, consider a diagnostic algorithm that returned the ranking $\langle c_{12}, c_5, c_6, \ldots \rangle$, while c_6 is the actual fault. In this case, the diagnosis leads the developer to inspect c_{12} and c_5 first. As both statements are healthy, C_d is increased with 2. The next statement to be inspected is c_6. As it is faulty, no more effort is wasted, and consequently, $C_d = 2$.

2.1 Statistical Fault Diagnosis

Statistical fault diagnosis is a well-known approach to fault diagnosis that originates from the Software Engineering domain. Its most known variant is Spectrum-based Fault Localization (SFL) [2, 11]. In SFL, the fault likelihood l_j is quantified in terms of *similarity coefficients*. A similarity coefficient (SC) measures the statistical similarity between statement c_j's test coverage $(a_{1j}, \ldots, a_{Nj})$ and the observed test outcomes, $(o_1, \ldots, o_N)$. Similarity is computed by means of four counters $n_{pq}(j)$ that count the number of times a_{ij} and o_i form the combinations $(0,0), \ldots, (1,1)$, respectively, i.e.,

$$n_{pq}(j) = |\{i \mid a_{ij} = p \wedge o_i = q\}| \quad p, q \in \{0, 1\} \tag{1}$$

For instance, $n_{10}(j)$ and $n_{11}(j)$ are the number of tests in which c_j is executed, and which passed or failed, respectively. The four counters sum up to the number of tests N. Two of the most known SCs are the Tarantula [11], and Ochiai [2] similarity coefficients, given by

$$\text{Tarantula:} \qquad l_j = \frac{\frac{n_{11(j)}}{n_{11(j)}+n_{01(j)}}}{\frac{n_{11(j)}}{n_{11(j)}+n_{01(j)}} + \frac{n_{10(j)}}{n_{10(j)}+n_{00(j)}}} \tag{2}$$

$$\text{Ochiai:} \qquad l_j = \frac{n_{11(j)}}{\sqrt{(n_{11(j)}+n_{01(j)}) \cdot (n_{11(j)}+n_{10(j)})}} \tag{3}$$

In our example system, after executing all the tests in the test suite, the counters for c_6 are $n_{11}(6) = 8$, $n_{10}(6) = 0$, $n_{01}(6) = 0$, $n_{00}(6) = 0$. Its likelihood being the faulty one according to Ochiai is $l_6 = 1.0$. The remaining statements have lower likelihoods, as they all have $n_{10}(j) > 0$ or $n_{01}(j) > 0$. For example, for c_5, $n_{11}(5) = 3$, $n_{10}(5) = 3$, $n_{01}(5) = 0$, $n_{00}(5) = 1$, and its Ochiai similarity is $l_5 = 0.71$.

Despite their lower diagnostic accuracy [4], similarity coefficients have gained much interest. A great advantage of similarity coefficients is their ultra-low computational complexity compared to probabilistic approaches and their independence from prior FNR knowledge. Therefore, we will include SC in our evaluation.

2.2 Bayesian Fault Diagnosis

Bayesian fault diagnosis is aimed at obtaining a set of fault candidates $D = \langle d_1, \ldots, d_k \rangle$. Each candidate d_k is a subset of the statements which, at fault, explain the observed failures. For instance, $d = \{c_1, c_3, c_4\}$ indicates that c_1 **and** c_3 **and** c_4 are faulty, and no other statement. These diagnostic candidates are again sorted by the probability of being the correct diagnosis, $\Pr(d_k)$. A more detailed description can be found in [6].

After test case t_i is executed, the probability of each candidate $d_k \in D$ is updated depending on the outcome o_i of the test, following Bayes' rule:

$$\Pr(d_k|o_i) = \frac{\Pr(o_i|d_k)}{\Pr(o_i)} \cdot \Pr(d_k) \tag{4}$$

which is repeated recursively for each executed test.

In this equation, $\Pr(d_k)$ represents the prior probability of candidate d_k before the test is executed. $\Pr(o_i)$ is a normalization value that represents the residual probability of the observed outcome, independent of which candidate is the actual diagnosis, and need not be computed directly. $\Pr(o_i|d_k)$ represents the probability of the observed outcome o_i produced by a test t_i, if that candidate d_k is the actual diagnosis. It is defined as

$$\Pr(o_i|d_k) = \begin{cases} 0 & \text{if } o_i \text{ and } d_k \text{ are inconsistent} \\ 1 & \text{if } o_i \text{ is unique to } d_k \\ \varepsilon(d_k) & \text{otherwise} \end{cases} \tag{5}$$

The value of ε is related directly to the false negative rate of the tests that cover the statements in d_k. Multiple policies for ε exist. In the next section we will describe the most common one.

3. ESTIMATING FNR

Correct information on the FNR of tests is crucial to a correct diagnosis, and its estimation has been a recurring research topic. The FNR of tests that cover the statements in a diagnosis candidate d_k depends on the FNR of every individual statement $c_j \in d_k$. The FNR of individual statements is modeled by a probability value $h_j \in [0, 1]$ (h for *health*). $h_j = 0$ means that when tested, a faulty statement will always produce a failure. $h_j = 1$ means that when tested, a faulty statement will never produce a failure.

The most common ε policy assumes that faulty statements will cause failures independently, e.g., a failure in a system with 2 faults will be produced when either of the faults causes a failure. This is known as the disjunctive failure model, i.e., "OR Model" [6].

The probability of a false negative occurring is, therefore, the probability of all faults not producing any failure. If the h_j values are known, they can be used to calculate this probability by

$$\varepsilon(d_k) = \begin{cases} \displaystyle\prod_{c_j \in d_k \wedge a_i j = 1} h_j & \text{if } o_i = 0 \\ 1 - \displaystyle\prod_{c_j \in d_k \wedge a_i j = 1} h_j & \text{if } o_i = 1 \end{cases} \tag{6}$$

For example, let's assume the h_j values for the system in Table 1 were known to be all 0.1, and we want to calculate the FNR of candidates $d_6 = \{c_6\}$ and $d_{8,10} = \{c_8, c_{10}\}$. In this case, for test t_1 ($o_1 = 1$), $\varepsilon(d_6) = 0.9$ and $\varepsilon(d_{8,10}) = 0.99$. For t_2 ($o_2 = 0$), $\varepsilon(d_6) = 1$ and $\varepsilon(d_{8,10}) = 0.1$. Assuming the initial priors are $\Pr(d_6) = \Pr(d_{8,10}) = 0.5$, the final updated value after executing all tests would be $\Pr(d_6) = 0.98$, $\Pr(d_{8,10}) = 0.02$.

3.1 Posterior FNR estimation

Unfortunately, the h_j values are hardly ever known *a priori*. A number of approximation strategies have been proposed for the case when the individual h_j are unknown. In this paper we will consider the $\varepsilon^{(2)}$ strategy [1] given by

$$\varepsilon^{(2)}(d_k) = \begin{cases} g(d_k)^\eta & \text{if } o_i = 0 \\ 1 - g(d_k)^\eta & \text{if } o_i = 1 \end{cases} \tag{7}$$

where η is the the number of faulty statements according to d_k involved in the test t_i, and $g(d_k)$ is the "effective" FNR parameter for d_k ,estimated by counting how many times statements in d_k are involved in failed and passed tests, according to

$$g(d_k) = \frac{\displaystyle\sum_{i=1}^{N} [(\bigvee_{j \in d_k} a_{ij} = 1) \wedge o_i = 0]}{\displaystyle\sum_{i=1}^{N} [\bigvee_{j \in d_k} a_{ij} = 1)]} \tag{8}$$

where $[\ldots]$ casts truth values into $[true] = 1$, $[false] = 0$.

The failures in the example system of Table 1 can be explained by candidate $d_6 = \{c_6\}$ and also by $d_{8,10} = \{c_8, c_{10}\}$. The value for $g(d_6) = \frac{0}{4}$, whereas $g(d_{8,10}) = \frac{2}{6}$. For the failing tests, the $\varepsilon(d_6) = 1 - 0$. In the case of $d_{8,10}$ ε depends on the number of statements involved. For t_1 $\varepsilon(d_{8,10}) = 1 - 0.33^2 = 0.88$, and for t_3, t_4, and t_7, $\varepsilon(d_{8,10}) = 1 - 0.33^1 = 0.66$ because there is only one statement involved. Assuming that $\Pr(d_6) = \Pr(d_{8,10}) = 0.5$ initially, after the update with all tests $\Pr(d_6) = 0.97$ and $\Pr(d_{8,10}) = 0.03$.

A stable estimation requires a large number of tests, and still may be imprecise. This reduces the rate at which the diagnosis improves, requiring the execution of a large number of tests to obtain a high quality diagnosis.

3.2 Prior FNR Estimation: Testability

False negatives in the context of software fault diagnosis are intimately connected with the concept of *testability*. Of the multiple definitions of testability, the one proposed by Voas and Miller [20] is directly related to the concept of false negatives: *the degree to which software reveals faults during testing*. Testability relates the number of tests to be executed on a faulty statement before it exposes the fault as a failure, i.e., the complement of the FNR, $1 - h$.

Testability can be estimated by the so-called propagation-infection-execution approach (PIE) [18]. The PIE approach involves an expensive mutation analysis. However, its implementation is simple and can be automated. In this paper, we use a variation of the PIE approach to obtain an estimation of the testability of a statement.

First, for each statement c_j for which we wish to calculate h_j, we created a set of mutants $\mathcal{M}_j$. This was done by applying a small set of mutation operators [15] to the arithmetic, logic and indexing operations contained in the statement or function. The mutations were done at the level of the bytecode representation used by the Zoltar [9] fault diagnosis tool. Second, the program's full test suite was run, recording test coverage and failure information for each of the mutants. Finally, the FNR of each statement c_j was obtained by averaging the FNR of each mutation in $\mathcal{M}_j$, calculated as the ratio of the number of tests which covered the fault but did not produce a failure, to the number of tests which covered the fault.

It is important to note that program mutation can produce equivalent mutants, i.e., altered programs but tested as being correct. These should be removed from the average.In our study, though, we kept those mutants, because the fact that a mutant did not produce any failure in the tests could also mean that there is no test case to render the mutant faulty. Therefore, the values obtained by our FNR estimation should be seen as pessimistic. This approach was also used in [17]. The cost of a testability scales with $O(k \cdot M \cdot N)$, where k is the average number of mutants generated per component.

4. EXPERIMENT SETUP

We will perform our experiments on the well-known Siemens benchmark set [8]. The Siemens set comprises seven programs, providing one correct version, and a set of faulty versions for each program. Every faulty version contains exactly one fault. As the few faults in Siemens would provide a very reduced set of fault combinations, in our experiments we use a larger set of faults obtained by random mutation, similar to the ones used in the PIE testability study. 100 faulty versions with 1 fault, and 100 faulty versions with 3 faults were created per program.

For each program, a test suite is provided to achieve full statement and branch coverage. As each program studied comes with a correct version, we use this as test oracle. Table 2 summarizes the programs used for empirical evaluation.

Following existing literature on fault diagnosis, we assume uniform *a priori* fault probability p_j [4]. The precise values of p_j have little influence on the performance of the diagnosis, as they are modified by the subsequent Bayesian update process.

SC		Tarantula	$l_j = \dfrac{\frac{n_{11(j)}}{n_{11(j)}+n_{01(j)}}}{\frac{n_{11(j)}}{n_{11(j)}+n_{01(j)}} + \frac{n_{10(j)}}{n_{10(j)}+n_{00(j)}}}$
		Ochiai	$l_j = \dfrac{n_{11(j)}}{\sqrt{(n_{11(j)}+n_{01(j)}) \cdot (n_{11(j)}+n_{10(j)})}}$
Bayes	Post	EPS2	$\varepsilon^{(2)}(d_k) = \begin{cases} g(d_k)^\eta & \text{if } o_i = 0 \\ 1 - g(d_k)^\eta & \text{if } o_i = 1 \end{cases}$
	Prior	PIE	$\varepsilon(d_k) = \begin{cases} \prod h_j & \text{if } o_i = 0 \\ 1 - \prod h_j & \text{if } o_i = 1 \end{cases}$
		SFB	$\varepsilon'(d_k) = \begin{cases} \varepsilon(d_k) & \text{if } \lvert d_k \rvert = 1 \\ 0 & \text{if } \lvert d_k \rvert > 1 \end{cases}$

Table 3: Overview of the compared techniques

Table 3 provides an overview of the compared techniques. For completeness we have added a fifth technique: single-fault Bayesian diagnostic with PIE information (SFB-Bayes), i.e., where only d_k containing one statement are considered ($\varepsilon(d_k) = 0$ if $|d_k| > 1$).

5. SINGLE FAULT EXPERIMENTS

The box plot in Figure 1 shows the performance of each technique in terms of C_d for all programs. In addition, Table 2 shows, the average C_d per program. The values have been normalized by dividing by the number of non-faulty statements $M - M_f$. From our results it is apparent how the introduction of *a priori* FNR information greatly improves the performance in the diagnosis.

SFB-Bayes provides the largest improvement possible, with reductions of diagnostic effort that can reach a 80% in the case of `schedule2` and are above 50% in many cases when compared with the Ochiai similarity coefficient. However, it must be taken into account that SFB-Bayes will produce an inconsistent diagnosis if more than one statement is faulty. For this reason its applicability is limited.

It can be seen how removing the single fault assumption poses a slight penalty in the improvements that can be achieved, which is due to the diagnosis algorithm considering equally probable multiple fault candidates. Improvements on diagnosis quality with PIE-Bayes can reach 72% for `schedule2` and 60% for `schedule`. However, for some programs this is not the case. For example `tot_info` does not show any improvement, and `print_tokens2` shows a performance degradation with respect to Ochiai.

The reason for the dramatic improvement of the quality of the diagnosis, especially in the case of SFB-Bayes, is the fact that PIE-Bayes is less affected by ambiguities in the diagnosis, i.e., uncertainty if there are different statements with equal likelihoods (due to their columns in A being identical). Similarity coefficients and EPS2-Bayes estimate identical likelihood values for statements with identical columns in A, making the diagnosis more ambiguous, leading to an increase in diagnosis effort. However, the h_j values estimated for the statements inside the same ambiguity group are typically not identical. Even if their execution patterns are identical, SFB and PIE-Bayes will rank first the statements whose h_j is closer to the real FNR of tests, effectively removing ambiguity.

6. MULTIPLE FAULT EXPERIMENTS

In this section we present the results of our experiments with programs seeded with multiple faults. Due to the small sizes of the programs in the Siemens set we seeded a maximum of $M_f = 3$ faults, in order to avoid unrealistic fault densities.

6.1 Diagnostic Performance

Once the validity of the OR model has been assessed, we will now comment on the performance of the multiple fault diagnosis techniques. It can be seen in Figure 1 and Table 2 how the introduction of PIE testability information greatly improves the quality of every diagnosis. PIE-Bayes provides improvements in diagnostic quality of an average of 37%, with `replace`, `schedule` and `schedule2` providing the largest improvements, of a 60%. On the bottom of the improvement scale is `tot_info` with just a 20%.

The reason for the improvements in the multiple fault case is similar to that for single faults. `tcas` and `tot_info` are the two programs with the highest approximation error on their FNR due to the failure independence assumption not holding, as we will explain in the following secion. This error is affecting the quality of the Bayesian update, reducing considerably the performance.

6.2 Testability and the OR Failure Model

Testability estimations obtained by the PIE method described in Section 3 are obtained in a single fault setting, i.e., the fault being studied is the only one existing in the system. However, when using PIE testability estimations in a *multiple-fault setting*, we must assess the error incurred when combining testability estimations by using the OR model as-

					$M_f = 1$					$M_f = 3$			
Program Name	Mutants	M	N	Program Type	SFB	PIE	EPS2	TAR	OCH	PIE	EPS2	TAR	OCH
`print_tokens`	491	539	4,130	Lexical Analyzer	0.038	0.064	0.100	0.248	0.077	0.210	0.422	0.473	0.397
`print_tokens2`	294	489	4,115	Lexical Analyzer	0.076	0.156	0.189	0.317	0.143	0.281	0.456	0.481	0.550
`replace`	757	507	5,542	Pattern Recognition	0.019	0.061	0.104	0.153	0.077	0.160	0.468	0.430	0.497
`schedule`	281	397	2,650	Priority Scheduler	0.024	0.041	0.120	0.244	0.097	0.188	0.549	0.459	0.630
`schedule2`	212	299	2,710	Priority Scheduler	0.072	0.094	0.362	0.490	0.343	0.211	0.604	0.630	0.565
`tcas`	208	174	1,608	Altitude Separation	0.104	0.134	0.211	0.230	0.197	0.313	0.481	0.451	0.477
`tot_info`	396	398	1,052	Information Measure	0.053	0.093	0.128	0.209	0.094	0.360	0.450	0.543	0.519

Table 2: Programs used for evaluation and results

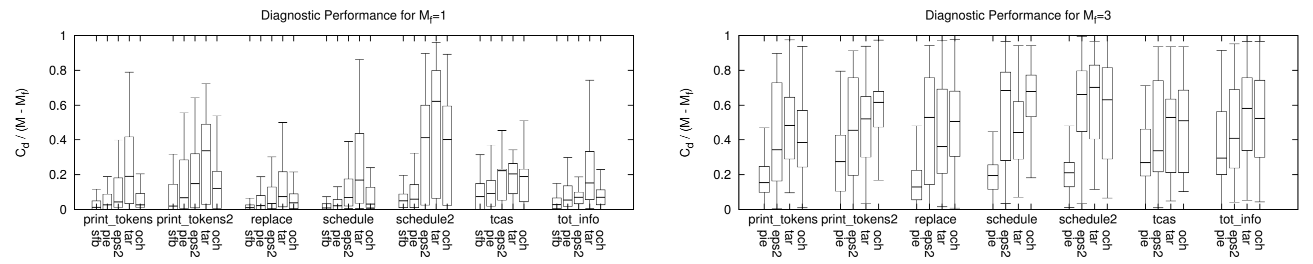

Figure 1: Performance for mutated programs

sumption in Equation 6, and whether failure independence holds in a multiple fault environment. This is crucial since Bayesian diagnosis relies on the assumption that failures are independent events and failure rates can be combined disjunctively.

In order to answer these questions, we measured the real FNR of each possible executed fault combination (F_1, F_2, F_3, $F_1 \wedge F_2$, $F_2 \wedge F_3$, $F_1 \wedge F_3$, $F_1 \wedge F_2 \wedge F_3$) occurring in the test matrix of our mutated programs, and compared their actual values with the value estimated with Equation 6 and PIE.

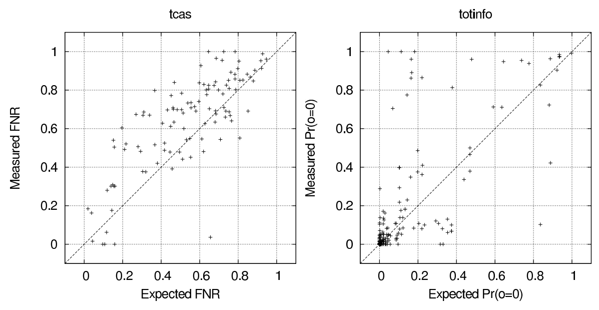

Figure 2: Estimation error of the FNR

Our experiment confirmed that, in general, the estimated FNR of a known combination of faults presents a moderate error due to the errors of the testability estimations used. However, in some cases, this error is extremely high. The scatter plots in Figure 2 show two examples of this situation. To explain these cases, one must remember that, in many cases, failures in software are heavily dependent on the distribution of failing inputs, which is abstracted away in fault diagnosis. The diagrams in Figure 3 present two situations in which the OR model and the failure independence assumption will hold, and two situations in which they will not. The circled area represents the inputs that cover a given fault, with passing inputs depicted as empty shapes, and failing inputs as filled shapes.

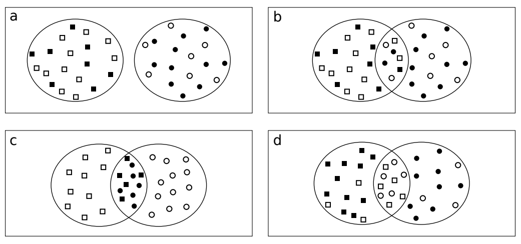

Figure 3: Weak OR-model at the input level

In case (a), the sets of inputs that cover each fault is disjoint. This case is frequent in software if the faults occur in parts of the software dedicated to very different tasks. Case (b) shows a case where the OR model is applied for two faults whose failures are uniformly distributed across the input domain. When combined, this uniform distribution will still hold for the intersection area, and the OR model will present a low estimation error.

The faults in cases (c) and (d), on the other hand, do not produce failures that are uniformly distributed across their input domains. In case (c), the OR model will lead to a failure probability that is too low for the $F1 \wedge F2$ combination, and too high for the $F1$ and $F2$ combinations. Case (d) presents the reverse situation.

7. THREATS TO VALIDITY

The Siemens programs used are small programs with very extensive and redundant test suites. The validity of our results can be strengthened by using larger, more realistic programs.

The validity of the results presented is further threatened by the fact that the same set of mutants were used to estimate h_j *and* to perform the diagnosis. Firstly, this could produce too optimistic results, as the estimated FNR corresponds exactly to the average of the actual FNR of the mutants being diagnosed. However, the seeded faults included in Siemens are, to a great extent, also generated by our mutation tool. Secondly, the density of the faults must be considered. As previously mentioned, we avoid performing our multiple fault experiments with more than 3 faults to keep the fault density at a realistic level. Finally, in our experiments we assumed that faults are distributed uniformly in the program code. The validity of our results could also be strengthened by considering the effect of fault distributions.

8. RELATED WORK

Automated fault localization techniques minimize diagnostic cost and support debugging when failures occur during software testing. Statistical approaches include [2, 11, 12, 13, 16, 21]. A recent, probabilistic approach of acceptable complexity is [4]. Although, different in the way they derive the ranking of the diagnosis, all techniques are based on measuring the coverage information and failure pattern of a program (also termed its spectrum), while ignoring or deriving the rate of false negatives a posteriori. The novelty of our approach consists in factoring the FNR estimation problem out of the diagnostic problem, so that FNR can be exploited from the beginning. False negatives in software are related to testability [20]. Testability quantification has been approached at the class level [5], function level [7] and statement level [14, 18, 19, 22, 23]. However, of the current state of the art, only [18] allows for a straightforward usage as FNR in our experiments. Our work is novel with respect to evaluating the degree to which software faults produce failures independently, a necessary assumption in fault diagnosis.

9. CONCLUSIONS AND FUTURE WORK

False negatives are an essential problem of fault localization. Previous work in diagnosis has considered FNR modeling a part of the diagnosis process. In this paper we have studied the improvement in software fault localization when information on the false negative rate is available *a priori*, by means of a PIE testability study.

PIE-Bayes reduces the average effort for coming to the diagnosis by a 27% for real programs with single faults in average when using multiple-fault techniques. When using SFB-Bayes the performance increases dramatically, with a 60% reduction of diagnostic effort on average.

For multiple faults, prior FNR information achieves a 37% reduction of diagnostic effort on average. The performance for the multiple fault case does not only depend on the quality of the FNR estimations, but also on the degree to which they are independent and can be combined meaningfully in a disjunctive fashion.

The improvement in the fault localization process was traded against an expensive testability analysis, although, it should be noted that, in practice, the cost of such analysis is amortized over many debug-repair cycles. Future work will investigate the usage of alternative testability estimation techniques, e.g. bridging the gap between static techniques and the probabilistic approach needed for PIE-Bayes.

10. REFERENCES

[1] R. Abreu, P. Zoeteweij, and A. van Gemund. An observation-based model for fault localization. In *Proc. WODA'08*.

[2] R. Abreu, P. Zoeteweij, and A. van Gemund. On the accuracy of spectrum-based fault localization. In *Proc. TAIC PART'07*.

[3] R. Abreu, P. Zoeteweij, and A. J. van Gemund. A new bayesian approach to multiple intermittent fault diagnosis. In *Proc. IJCAI'09*.

[4] R. Abreu, P. Zoeteweij, and A. J. van Gemund. Spectrum-based multiple fault localization. In *Proc. ASE'09*.

[5] M. Bruntink and A. van Deursen. An empirical study into class testability. *J. Syst. Softw.*, 79(9):1219–1232, 2006.

[6] J. de Kleer. Diagnosing multiple persistent and intermittent faults. In *Proc. IJCAI'09*.

[7] R. S. Freedman. Testability of software components. *IEEE TSE*, 17(6):553–564, 1991.

[8] M. Hutchins, H. Foster, T. Goradia, and T. Ostrand. Experiments of the effectiveness of dataflow- and controlflow-based test adequacy criteria. In *Proc. ICSE'94*.

[9] T. Janssen, R. Abreu, and A. J. C. van Gemund. Zoltar: A toolset for automatic fault localization. In *Proc. ASE'09 - Tool Demonstrations*.

[10] B. Jiang, Z. Zhang, W. Chan, and T. Tse. Adaptive random test case prioritization. In *Proc. ASE'09*.

[11] J. A. Jones, M. J. Harrold, and J. Stasko. Visualization of test information to assist fault localization. In *Proc. ICSE'02*.

[12] B. Liblit. Cooperative debugging with five hundred million test cases. In *Proc. ISSTA'08*.

[13] C. Liu, X. Yan, L. Fei, J. Han, and S. P. Midkiff. Sober: Statistical model-based bug localization. In *Proc. ESEC/FSE-13*.

[14] A. J. Offutt and J. H. Hayes. A semantic model of program faults. *SIGSOFT Soft. Eng. Notes*, 21(3):195–200, 1996.

[15] A. J. Offutt, A. Lee, G. Rothermel, R. H. Untch, and C. Zapf. An experimental determination of sufficient mutant operators. *ACM TOSEM*, 5:99–118, April 1996.

[16] M. Renieris and S. P. Reiss. Fault localization with nearest neighbor queries. In *Proc. ASE'03*.

[17] G. Rothermel, R. H. Untch, C. Chu, and M. J. Harrold. Prioritizing test cases for regression testing. *IEEE TSE*, 27(10), 2001.

[18] J. M. Voas. Pie: A dynamic failure-based technique. *IEEE TSE*, 18(8):717–727, 1992.

[19] J. M. Voas and K. W. Miller. Semantic metrics for software testability. *Journal of Systems and Software*, 20(3):207–216, 1993.

[20] J. M. Voas and K. W. Miller. Software testability: The new verification. *IEEE Software*, 12(3):17–28, 1995.

[21] W. Wong, T. Wei, Y. Qi, and L. Zhao. A crosstab-based statistical method for effective fault localization. In *Proc. ICST'08*.

[22] M. R. Woodward and Z. A. Al-Khanjari. Testability, fault size and the domain-to-range ratio: An eternal triangle. *SIGSOFT Soft. Eng. Notes*, 25(5):168–172, 2000.

[23] L. Zhao. A new approach for software testability analysis. In *ICSE '06*.