Localizing Software Faults Simultaneously[⋆]

Rui Abreu and Peter Zoeteweij and Arjan J.C. van Gemund
Embedded Software Lab
Faculty Electrical Eng., Mathematics, and Computer Science
Delft University of Technology
The Netherlands
Email: {r.f.abreu, p.zoeteweij, a.j.c.vangemund}@tudelft.nl

***Abstract*—Current automatic diagnosis techniques are predominantly of a statistical nature and, despite typical defect densities, do not explicitly consider multiple faults, as also demonstrated by the popularity of the single-fault Siemens set. We present a logic reasoning approach, called Zoltar-M(ultiple fault), that yields multiple-fault diagnoses, ranked in order of their probability. Although application of Zoltar-M to programs with many faults requires further research into heuristics to reduce computational complexity, theory as well as experiments on synthetic program models and two multiple-fault program versions from the Siemens set show that for multiple-fault programs this approach can outperform statistical techniques, notably spectrum-based fault localization (SFL). As a side-effect of this research, we present a new SFL variant, called Zoltar-S(ingle fault), that is provably optimal for single-fault programs, outperforming all other variants known to date.**

***Keywords*-Software fault diagnosis, program spectra, statistical and reasoning approaches.**

I. Introduction

Automatic fault localization techniques aid developers to pinpoint the root cause of software faults, thereby reducing the debugging effort. Two approaches can be distinguished, (1) the *spectrum-based fault localization* (SFL) approach, which correlates software component activity with program failures (a *statistical* approach) [2], [7], [10], [11], [16], [19], and (2) the *model-based diagnosis* or *debugging* (MBD) approach, which deduces component failure through *logic reasoning* [5], [6], [13], [18].

Because of its low computational complexity, SFL has gained large popularity. Although inherently not restricted to single faults, in most cases these statistical techniques are applied and evaluated in a single-fault context, as demonstrated by the Siemens benchmark set which is seeded with only 1 fault per program (version). In practice however, the defect density of even small programs typically amounts to multiple faults. Although the root cause of a particular program failure need not constitute multiple faults that are acting *simultaneously*, many failures will be caused by *different* faults. Hence, the problem of *multiple-fault localization* (diagnosis) deserves detailed study.

Unlike SFL, MBD traditionally deals with multiple faults. However, apart from much higher computational complexity, the logic models that are used in the diagnostic inference are typically based on static program analysis. Consequently, they don't exploit execution behavior, which, in contrast, is the essence of the SFL approach. Combining the dynamic approach of SFL with the multiple-fault logic reasoning approach of MBD, in this paper we present a multiple-fault reasoning approach that is based on the dynamic, spectrum-based observations of SFL. Additional reasons to study the merits of this approach are the following. (1) Diagnoses are returned in terms of multiple faults, whereas statistical techniques return a one-dimensional list of single fault locations only. The information on fault multiplicity is attractive from parallel debugging point of view [9]. (2) Unlike statistical approaches, multiple-fault diagnoses only include valid candidates, and are asymptotically optimal with increasing test information [1]. (3) The ranking of the diagnoses is based on probability instead of similarity. This implies that the quality of a diagnosis can be expressed in terms of information entropy or any other metric that is based on probability theory [14]. (4) The reasoning approach naturally accommodates additional (model) information about component behavior, increasing diagnostic performance when more information about component behavior is available.

To illustrate the difference between multiple-fault and the statistical approach, consider a triple-fault (sub)program with faulty components c_1, c_2, and c_3. Whereas under ideal testing circumstances a traditional SFL approach would produce multiple single-fault diagnoses (in terms of the component indices) like $\{\{1\}, \{2\}, \{3\}, \{4\}, \{5\}, \ldots\}$ (ordered in terms of statistical similarity), a multiple-fault approach would simply produce one single multiple-fault diagnosis $\{\{1, 2, 3\}\}$. Although the statistical similarity of the first three items in the former diagnosis would be highest, the latter, single diagnosis unambiguously reveals the actual triple fault.

Despite the above advantages, a reasoning approach is more costly than statistical approaches because an exponential number of multiple-fault candidates need to be processed

[⋆]This work has been carried out as part of the TRADER project under the responsibility of the Embedded Systems Institute. This project is partially supported by the Netherlands Ministry of Economic Affairs under the BSIK03021 program.

instead of just the single-fault candidates. In this paper we compare our reasoning approach to several statistical approaches. Our study is based on random synthetic spectra, as well as on the Siemens benchmark, extended by us to accommodate multiple faults. More specifically, this paper makes the following 5 contributions. (1) We introduce a multiple-fault diagnosis approach that originates from the model-based diagnosis area, but which is specifically adapted to the interaction dynamics of software. The approach is coined Zoltar-M (Zoltar for the name of our debugging tool set, M for multiple-fault). (2) We show how our reasoning approach applies to single-fault programs, yielding a provably optimal SFL variant, called Zoltar-S (S for single-fault), as of yet unknown in literature. (3) We introduce a general, multiple-fault, probabilistic program (spectrum) model, parametrized in terms of size, testing code coverage, and testing fault coverage, to theoretically study Zoltar-M, compared to statistical techniques such as Tarantula and Zoltar-S. (4) We investigate the ability of all techniques to deduce program fault multiplicity which is aimed at providing a good estimate to guide parallel debugging, using an approach that substantially differs from [9].

To the best of our knowledge, this is the first paper to specifically address software multiple-fault localization using a spectrum-based, logic reasoning approach, yielding two new localization techniques Zoltar-S and Zoltar-M, implemented within our Zoltar SFL framework. Our experiments confirm that Zoltar-S is superior to all known similarity coefficients for the Siemens-S benchmark. More importantly however, our experiments for multiple-fault programs show that although for synthetic spectra Zoltar-M is outperformed by Zoltar-S, for our Siemens-M experiments Zoltar-M outperforms all similarity coefficients known to date.

II. Preliminaries

In this section we introduce basic definitions as well as the traditional SFL approach.

A. Basic Definitions

A program that is being diagnosed comprises a set of M components (statements in the context of this paper), which is executed using N test cases that either pass of fail. Program (component) activity is recorded in terms of program spectra [8]. This data is collected at run-time, and typically consists of a number of counters or flags for the different components of a program. In the context of this paper we use the so-called hit spectra, which indicate whether a component was involved in a (test) run or not.

Both spectra and pass/fail information is input to traditional SFL, as well as to our reasoning technique. The combined information is expressed in terms of the $N \times (M+1)$ *observation matrix* O. An element o_{ij} is equal to 1 if component j was observed to be *involved* in the execution of run i, and 0 otherwise. The element $o_{i,m+1}$ is equal to 1 if run i *failed*, and 0 if run i *passed*. The rightmost column of O is also denoted as e (the error vector). From O we can derive the probability r that a component is actually executed in a run (testing code coverage), and the probability g that a faulty component is actually exhibiting good behavior (testing fault coverage, also known as the "goodness" parameter g from MBD [3]).

Programs can have multiple faults, the number being denoted C (fault cardinality). A *diagnosis candidate* is expressed as the set of indices of those components whose *combined* faulty behavior is logically consistent with the observations O and therefore must be considered as a collective candidate. A *diagnosis* is the ordered set of diagnostic candidates $D = \{d_1, \ldots, d_k\}$, all of which are an explanation consistent with observed program behavior (O), ordered in probability of being the program's actual multiple fault condition. An example multiple-fault diagnosis is the diagnosis $\{d_1\} = \{\{1,2,3\}\}$ given in the Introduction. For brevity, we will often refer to diagnostic candidates as diagnoses as well, as it is clear from the context whether we refer to a single diagnosis candidate or to the entire diagnosis.

B. Traditional SFL

In SFL one measures the similarity between the error vector e and the activity profile vector O_{*j} for each component j. This similarity is quantified by a *similarity coefficient*, expressed in terms of four counters $a_{pq}(j)$ that count the number of positions in which O_{*j} and e contain respective values p and q, i.e, for $p, q \in \{0,1\}$, we define $a_{pq}(j) = |\{i \mid o_{ij} = p \wedge e_i = q\}|$. Two examples of well-known coefficients are

$$s_T = \frac{\frac{a_{11}(j)}{a_{11}(j)+a_{01}(j)}}{\frac{a_{11}(j)}{a_{11}(j)+a_{01}(j)} + \frac{a_{10}(j)}{a_{10}(j)+a_{00}(j)}}$$

as used by the Tarantula tool [10], and the Ochiai coefficient

$$s_O = \frac{a_{11}(j)}{\sqrt{(a_{11}(j)+a_{01}(j)) * (a_{11}(j)+a_{10}(j))}} \qquad (1)$$

known from molecular biology, introduced in SFL in [2].

As an example, suppose we have a program with $M = 7$ components, of which c_1, c_2, and c_3 are faulty, with O as given in Table I. The table also includes the a_{pq} counts as well as the resulting similarity based on the Tarantula and Ochiai coefficients. Assuming that a developer would follow the ranking produced by the techniques, Tarantula requires him/her to inspect more components in order to find a faulty one. The first faulty component ranked by Tarantula is at the 3rd place of the list, whereas with Ochiai it is already at the 2nd place. The results shows the sensitivity of Tarantula to components that are not involved in passed runs (a_{00}), considering them likely to be the faulty one and not taking

	c_1	c_2	c_3	c_4	c_5	c_6	c_7	e
	1	0	1	0	1	0	0	0
	0	0	0	0	1	0	0	0
	0	0	1	0	0	1	0	0
	1	1	0	0	1	0	0	0
	1	0	1	0	0	0	0	0
	1	0	0	0	1	0	0	0
	1	0	1	0	0	1	0	0
	1	1	0	0	1	1	0	1
	1	1	0	1	0	1	1	1
	1	0	1	0	0	0	0	1
	1	1	0	1	1	1	1	1
	1	1	0	0	0	1	0	1
	1	0	1	0	0	1	1	1
s_T	0.58	0.82	0.37	1	0.37	0.74	1	
s_O	0.74	0.73	0.33	0.58	0.33	0.77	0.71	

Table I
OBSERVATION MATRIX EXAMPLE O

into account their involvement in failed runs (e.g., c_4 and c_7). Ochiai, however, exonerates components based on their involvement in passed runs (a_{10}), and absence in failed runs (a_{01}, for detailed comparison, see [2]).

As can be seen, both Tarantula and Ochiai fail to consider c_3 as one of the most suspicious components. Besides, c_2 and c_3 can be considered as multiple fault because all failed runs can be explained either by c_2 or c_3 (but the two by themselves are not a valid explanation for all failures). In the next section we present our technique that exploits this info and contains multiple-fault explanations in its ranking.

III. MULTIPLE-FAULT LOCALIZATION

In this section we present our multiple-fault localization approach Zoltar-M, which is based on logic reasoning as performed in model-based diagnosis, combined with (Bayesian) probability theory to compute the ranking of the candidates. The major difference with the statistical approach in Section II-B is (1) that only a *subset* of components is considered (the so-called hitting set) in contrast to all components, (2) all computed candidates logically explain the observed failures, and (3) that the ranking is based on probability, rather than statistical similarity.

A. Hitting Set Computation

In model-based diagnosis one derives a model of the program, which, together with the observations of input-output behavior determine the set of constraints from which diagnostic solutions are logically deduced that are consistent with this behavior. Unlike the MBD approaches such as presented in [12], [13], in our Zoltar-M approach we refrain from modeling the program in detail, but use O as the only, dynamic source of information, from which we derive the model and the input-output observations.

Each component c_j is modeled by the logic proposition

$$h_j \Rightarrow in.ok_j \Rightarrow out.ok_j \qquad (2)$$

where h_j models the health state of c_j (*true* = healthy, *false* = defect). This so-called *weak* model specifies that a component produces correct output values ($out.ok$ true) if (1) healthy (h is true), and (2) when provided with correct input ($in.ok$ true). Note that this model still allows a component to produce correct data (the probability of which is measured by g) even though $h = false$. Also note that a component can accept erroneous input data and still produce correct output. Finally, this approach allows the inclusion of additional component information as the number of propositions per component is not limited to the above, default model of nominal behavior.

Due to dynamic (data-dependent) control flow each run may involve different components. Consequently, rather than modeling the program by a static composition of component propositions (Eq. 2), we consider a dynamic model that is defined per program run (i.e., O_{i*}). In [1] it is shown that each failed run yields a conjunction of logical constraints (i.e., a sub-model) in terms of the components involved, that is known in MBD as a *conflict* [4]. For instance, a failed run involving c_1 and c_2 generates the conflict $\neg h_1 \vee \neg h_2$, indicating that c_1 and c_2 cannot both be healthy.

The multiple-fault approach is based on compiling each failed run (row O_{i*}) to a conflict, after which the diagnosis for O is derived by computing the hitting set [15] from all conflicts [1] (the hitting set algorithm essentially transforms logic products-of-sums into sums-of-products). For instance, the example observation matrix O in Table I generates the following 6 conflicts

$$\begin{aligned}
&(\neg h_1 \vee \neg h_2 \vee \neg h_5 \vee \neg h_6) \wedge \\
&(\neg h_1 \vee \neg h_2 \vee \neg h_4 \vee \neg h_6 \vee \neg h_7) \wedge \\
&(\neg h_1 \vee \neg h_3) \wedge \\
&(\neg h_1 \vee \neg h_2 \vee \neg h_4 \vee \neg h_5 \vee \neg h_6 \vee \neg h_7) \wedge \\
&(\neg h_1 \vee \neg h_2 \vee \neg h_6) \wedge \\
&(\neg h_1 \vee \neg h_3 \vee \neg h_6 \vee \neg h_7)
\end{aligned}$$

The (minimal) hitting set comprises one single-fault candidate $\{1\}$, and two double-fault candidates $\{2,3\}$ and $\{3,6\}$. Note that the triple-fault candidate $\{1,2,3\}$, which equals the actual fault state, is subsumed by both $\{1\}$ and $\{2,3\}$ and therefore does not appear in D (which is the *minimal* hitting set). The reason why, e.g., $\{1\}$ subsumes $\{1,j\}, j = 2,3,\ldots,M$ is that the weak component model (2) allows any faulty component j to exhibit correct behavior. Hence $\{1,j\}$ is also a valid explanation. The hitting set can be directly observed from O by (multi-)column "chains" of '1's from top to bottom formed by all failing rows of O. Note that this procedure only considers failed runs. Passed runs are considered later on when computing the probability of each diagnostic candidate.

B. Probability Computation

For each multiple-fault candidate the probability of being the actual diagnosis depends on the extent to which that candidate explains all observations (pass or fail per run). Let $\Pr(\{j\})$ denote the *a priori* probability that a component c_j is at fault. Although this value is typically dependent on code complexity, design, etc., we will simply assume $\Pr(\{j\}) = p$ (we arbitrarily set $p = 0.01$ in the context of this paper). Assuming components fail independently, and in absence of any observation, the prior probability a particular diagnosis d_k is correct is given by $\Pr(d_k) = p^{|d_k|} \cdot (1-p)^{M-|d_k|}$. Similar to the incremental compilation of conflicts per run we compute the posterior probability for each candidate based on the pass/fail observation obs for each sequential run using Bayes' update rule according to

$$\Pr(d_k|obs) = \frac{\Pr(obs|d_k)}{\Pr(obs)} \cdot \Pr(d_k)$$

The denominator $\Pr(obs)$ is a normalizing term that is identical for all d_k and thus needs not to be computed directly. $\Pr(obs|d_k)$ is defined as

$$\Pr(obs|d_k) = \begin{cases} 0 & \text{if } d_k \text{ and } obs \text{ are inconsistent} \\ 1 & \text{if } d_k \text{ logically follows from } obs \\ \epsilon & \text{if neither holds} \end{cases}$$

In the context of model-based diagnosis, many policies exist for ϵ (see [3]). In this paper we define ϵ as follows

$$\epsilon = \begin{cases} g(d_k)^t & \text{if run passed} \\ 1 - g(d_k)^t & \text{if run failed} \end{cases}$$

where t is the number of faulty components involved in the run (the rationale being that the more faulty components are involved, the more likely it is that the run will fail [1]), and where g is estimated by

$$g(d_k) = \frac{a_{10}(d_k)}{a_{10}(d_k) + a_{11}(d_k)}$$

where $a_{1q}(d_k) = \sum_{i=1..N} [(\bigvee_{j \in d_k} o_{ij} = 1) \wedge e_i = q]$ is a generalization of the definition in Section II-B to support multiple fault explanations, $q \in \{0, 1\}$, and $[\cdot]$ denotes Iverson's operator($[\texttt{true}] = 1]$, $[\texttt{false}] = 0]$).

To illustrate the differences between the probabilistic approach as presented in this section and the statistical SFL approach (as explained in Section II-B), again consider the example O in Table I. The diagnosis D for the different approaches are listed in Table II. As can be seen, the top ranked candidate for both Tarantula and Ochiai is not one of the three faulty locations, whereas for Zoltar-M one of the faults, namely c_1 would be immediately found. Furthermore, in contrast to Zoltar-M which contains multiple faults explanations such as $\{2, 3\}$, Tarantula and Ochiai only rank single-fault explanations. To conclude, note that Zoltar-M *only* lists candidates that actually *explain* all observed failures.

Technique	$D = \{d_1(s\|pr), \ldots, d_k(s\|pr)\}$
Tarantula	$\{\{4\}(1), \{7\}(1), \{2\}(0.82), \{6\}(0.74), \{1\}(0.58), \{3\}(0.37), \{5\}(0.37)\}$
Ochiai	$\{\{6\}(0.77), \{1\}(0.74), \{2\}(0.73), \{7\}(0.71), \{4\}(0.58), \{3\}(0.33), \{5\}(0.33)\}$
Zoltar-M	$\{\{1\}(0.98), \{2,3\}(0.99e^{-2}), \{3,6\}(0.52e^{-2})\}$

Table II
DIAGNOSES FOR EXAMPLE O

While the inherent multiple-fault approach used in Zoltar-M is asymptotically optimal, the complexity of the underlying hitting set algorithm and subsequently having to manage a possibly exponential number of multiple-fault candidates (e.g., update their probability) is prohibitive for large C (and N, M). Nevertheless, preliminary experiments with a statistically directed search technique (i.e., using statistical similarity to guide the search) indicates that the complexity of our current hitting set computation can be reduced by several orders of magnitude. In addition, hitting set completeness can be traded-off to further reduce time complexity (see [6] for a greedy stochastic search addressing this issue).

C. Single-fault Case

In this section we show how our above reasoning approach can be used to derive an optimal similarity coefficient for *single-fault* programs.

In the single-fault case (such as the Siemens-S benchmark) we know that all failures relate to only one fault, which, by definition, is included in the minimal hitting set. Hence, any coefficient approach should consider the minimal hitting set only (i.e., only those c_j which consistently occur in failing runs). Since for these components by definition $a_{01} = 0$, one only needs to consider a_{11} and a_{10}. This, in turn, implies that the ranking is only determined by the exonerating term a_{10}. In summary, once we only consider the components included in the hitting set, any of the coefficients that includes a_{10} in the denominator will produce the same, optimal ranking. Experiments using this "hitting set filter" combined with a simple similarity coefficient such as Tarantula indeed confirm that this approach leads to the best performance [17].

Note that the above filter is only optimal for programs that have only 1 fault as applying this filter to any multiple-fault program would be overly restrictive. It would fail to detect faults that are not always involved in failed runs. For example, the diagnosis for the O in Table I when using the filtering approach would yield $D = \{\{1\}\}$, entirely ignoring two of the three faults. Hence, instead of considering a single-fault hitting set filter, we modify this approach in order to also allow application to multiple-fault programs. Taking the Ochiai coefficient as (best) starting point (for $\kappa = 1$, Eq. 3 follows from Eq. 1 by squaring, and factoring

Program	Faulty Versions	M	N	Description
`print_tokens`	7	539	4,130	Lexical Analyzer
`print_tokens2`	10	489	4,115	Lexical Analyzer
`replace`	32	507	5,542	Pattern Recognition
`schedule`	9	397	2,650	Priority Scheduler
`schedule2`	10	299	2,710	Priority Scheduler
`tcas`	41	174	1,608	Altitude Separation
`tot_info`	23	398	1,052	Information Measure

Table III
THE SIEMENS BENCHMARK SET

out $a_{11}(j)$, none of which changes the ranking) and applying the above filtering approach, we derive the following similarity coefficient [1], coined Zoltar-S, according to

$$s_{\text{Z-S}} = \frac{a_{11}(j)}{a_{11}(j) + a_{10}(j) + a_{01}(j) + \kappa \cdot \frac{a_{01}(j) \cdot a_{10}(j)}{a_{11}(j)}} \quad (3)$$

where $\kappa > 0$ is a constant factor that exonerates a component c_j that was either seldom executed in failed runs or often in passed runs. We empirically verified that the higher the κ the more identical the diagnosis becomes with the one obtained by the hitting set filter [17]. In the context of this paper we limit κ to $10,000$ to avoid round-off errors.

To evaluate the diagnostic capabilities of Zoltar-S in comparison with other techniques, the Siemens-S set is used. This well-known benchmark is composed of 7 programs (see Table III; for detailed info, visit http://sir.unl.edu). In total Siemens-S provides 132 faulty programs. However, as no failures are observed in two of these programs, namely version 9 of `schedule2` and version 32 of `replace`, they are discarded. Besides, we also discard versions 4 and 6 of `print_tokens` because the faults are not in the program itself but in a header file. In summary, we discarded 4 versions out of 132 provided by the suite, using 128 versions in our experiments. To collect the program spectra, the `gcov`[1] profiling tool was used. For compatibility with previous work in (single-) fault localization, we use the effort/*s*core metric [2], [16] which is the percentage of statements that need to be inspected to find the fault - in other words, the rank position of the faulty statement divided by the total number of statements. Note that some techniques such as in [11], [16] do not rank all statements in the code, and their rankings are therefore based on the PDG of the program.

Figure 1 plots the percentage of located faults in terms of debugging effort [2]. Apart from the coefficients studied for SFL, the following techniques are also plotted: Intersection and Union [16], Delta Debugging (DD) [19], Nearest Neighbor (NN) [16], and Sober [11], which are among the best SFL techniques (detailed discussion in Section VI). As Sober is publicly available, we run it in our own environment. The values for the other techniques are, however, directly cited from their respective papers.

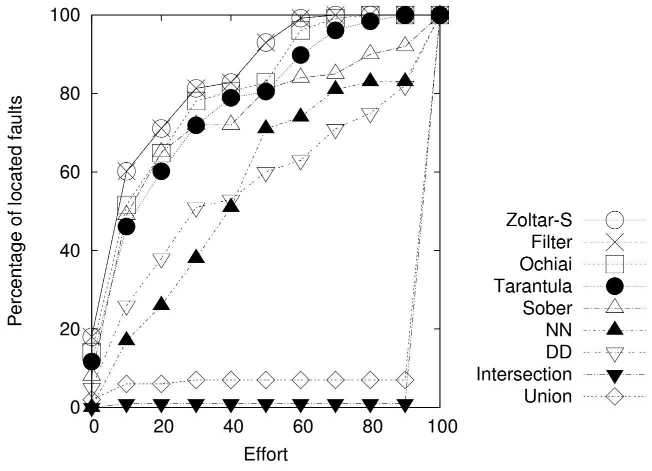


Figure 1. Effectiveness Comparison ($C = 1$)

From Figure 1, we conclude that Zoltar-S and the filter version are consistently the best performing techniques (note that in the single-fault context Zoltar-M simply reduces to Zoltar-S), finding 60% of the faults by examining less than 10% of the source code. For the same effort, using Ochiai would lead a developer to find 52% of the faulty versions and with Tarantula only 46% would be found. The Zoltar-S approach is followed by Ochiai, which outperforms Sober and Tarantula, which as concluded in [11], yield similar performance. Finally, the other techniques plotted are clearly outperformed by the spectrum-based techniques.

IV. THEORETICAL EVALUATION

In order to gain understanding of the effects of the various parameters on the diagnostic performance of the different approaches, we use a simple, probabilistic model of program behavior that is directly based on C, N, M, r, and g. Without loss of generality we model the first C of the M components to be at fault. For each run each component has probability r to be involved in that run. If a selected component is faulty, the probability of exhibiting nominal ("good") behavior equals g. When either of the C components fails, the run will fail. We study the performance of Zoltar-M in comparison to Tarantula, Ochiai, and Zoltar-S for observation matrices that are randomly generated according to the above model.

A. Performance Metrics

Before evaluating the results, we first present our performance metric. As one of the motivations of our multiple-fault approach is the exposure of fault multiplicity (parallel debugging) we refrain from reusing established metrics such as the diagnostic quality [2] or score [16] but evaluate the amount of *wasted debugging effort* W as a function of the number of parallel debuggers [9], denoted by P which

[1]http://gcc.gnu.org/onlinedocs/gcc/Gcov.html

P	1	2	3	4	5	6	...
Tarantula W/I	14/0	29/0	29/1	43/1	43/2	43/3	...
Zoltar-M W/I	0/1	0/2	0/3	14/3	–	–	...

Table IV
WASTED EFFORT FOR DIFFERENT DEVELOPERS P

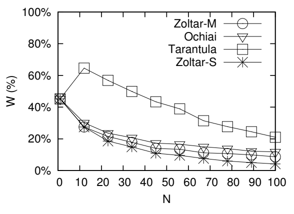
more clearly indicates practical debugging parallelism. The wasted debugging effort is computed as follows. From the diagnosis (obtained with either a statistical or reasoning approach) the first P candidates are examined (debugged) in parallel [9]. Actual faults are assumed to be properly debugged, after which the program is retested. Based on the retest a new diagnosis is obtained (excluding the repaired components, but including the still uncovered faults that may have considerably moved up in the ranking). This P-parallel process continues until in the last iteration the program retests ok (i.e., all faults have been found). W measures the percentage of non-faulty components that were debugged in the above process. For $P = 1$ the above procedure reduces to a standard sequential debugging process. For example, consider the diagnostic reports yield by Tarantula and Zoltar-M (as in Table II) for the example O in Table I. Table IV shows the performance profile for these two techniques (I stands for the number of bugs found in the first debugging iteration). As can be seen, Tarantula would need more developers in order to get a bug-free program in one iteration (6 developers against 3 for Zoltar-M). Furthermore, for this example, the wasted effort is consistently higher for Tarantula: with Zoltar-M, 3 developers would eliminate all bugs from the program at the cost of 0% wasted effort, whereas with Tarantula 6 developers would be needed at a cost of 43%. Note that there is no point in putting more than 4 developers to work as the Zoltar-M diagnosis contains only 4 different components.

Another reason not to adopt the aforementioned score metric [16] is that in our synthetic model we do not have program dependence graph information. Furthermore, the choice to exclude the actual faults from the debugging effort (i.e., instead of counting them as effort) is to make our performance metric independent of the number of faults C.

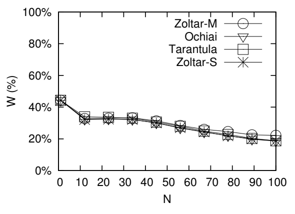
B. Experimental Results

In our first experiment we focus on the effect of C, N, M, r, and g on W. Consequently, we choose $P = 1$. We have varied M between 10 and 30 and after verifying that this does not change our conclusions [1], we choose $M = 20$ for the plots in the paper. Similarly, we also varied r between $r = 0.4$ and $r = 0.6$, and as there are no significant differences we only include the plots for $r = 0.6$, which is roughly the same as the values measured for the Siemens set.

Figure 2 plot W versus N for $C = 5$. We have also applied the technique for matrices with $C = 1$, $C = 2$



(a) $g = 0.1$

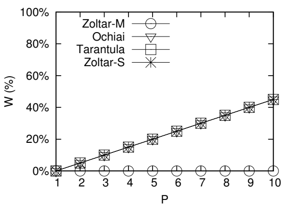


(b) $g = 0.9$

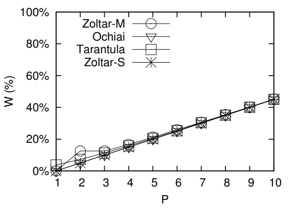
Figure 2. Wasted effort W for $C = 5$

and $C = 8$ and the conclusions are essentially the same. Each measurement represents an average over 1,000 sample matrices. The plots show that for small N all techniques start with equal W (for $N = 1$ it follows that $W = (M - C) \cdot r/M$ [1]), while for sufficiently large N all techniques produce an optimal diagnosis. The plots clearly show that all techniques yield an optimal diagnosis for sufficiently large N. This happens earlier for small C and g. In the single-fault case there is hardly any difference in the various techniques. For small g almost each run that involves the faulty component yields a failure, already producing near-perfect diagnoses for only small N. For large g (which is more realistic, the Siemens set exhibits g values ranging from 79% (`tot_info`) to 99% (`tcas`)), the fraction of failing runs dramatically decreases (cf. f in Section III-A). Consequently, a much higher number of runs is required to obtain a good diagnosis. For $C = 5$, we see the same trend, albeit that convergence to good diagnosis is much slower, especially for high g. This is due to the (combinatorial) fact that the number of "competitor" candidates of cardinality $B \leq C$ (see Section 3.1) greatly increases with C (and M [1]). The main conclusions is that all techniques are very similar for the synthetic matrices and no technique clearly outperforms the other. For $C = 5$ we conclude that for small g Zoltar-S outperforms all other techniques, Zoltar-M is the second-best technique, and Tarantula is the worst performing technique. Besides, for small g and N, the Tarantula technique has very poor performance because some of the non-faulty components are only touched in failed runs, hence sharing the first position of the ranking and degrading the diagnosis. This is also the reason why the wasted effort first increases and only then starts to decrease (more passed runs are needed for non-faulty components to be exonerated). The fact that Zoltar-S outperforms Zoltar-M comes from the fact that for the synthetic matrices there are not that many non-faulty components involved in all failed runs, and therefore Zoltar-S manages to rank the faulty components on top. For more realistic cases ($g = 0.9$) all techniques perform equally (poor), and much higher N is required to produce high diagnostic quality.

In Figures 3 and 4 we measure W for all approaches as function of P to study inherent debugging parallelism. For these plots we set $N = 500$ to ensure that each technique has

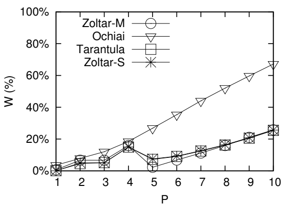


(a) $g = 0.1$

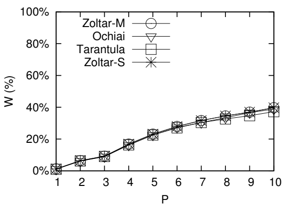


(b) $g = 0.9$

Figure 3. W vs. P for $C = 1$

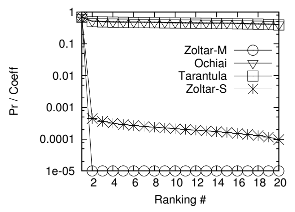


(a) $g = 0.1$

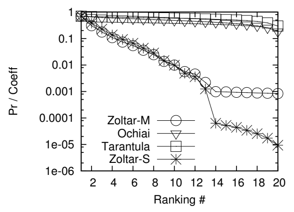


(b) $g = 0.9$

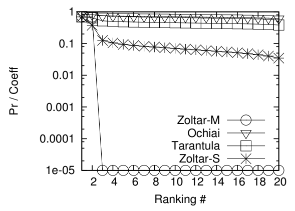
Figure 4. W vs. P for $C = 5$



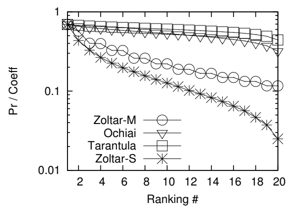
(a) $g = 0.1$ and $C = 1$



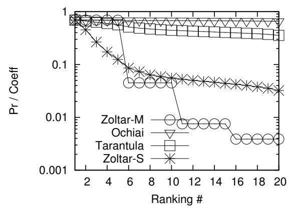
(b) $g = 0.9$ and $C = 1$



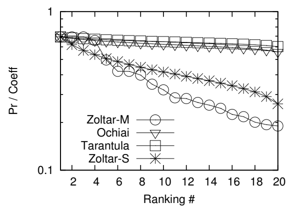
(c) $g = 0.1$ and $C = 2$



(d) $g = 0.9$ and $C = 2$



(e) $g = 0.1$ and $C = 5$



(f) $g = 0.9$ and $C = 5$

Figure 5. Probability/Similarity distribution

reached acceptable diagnostic quality. For $g = 0.1$, W starts a linear increase after $P = C$ (the "knee"), which indicates that all C faults are indeed at or near the top of the ranking (the bump at $P = 4$ is due to integer division effects). Except for Ochiai, both Zoltar-S and Tarantula yield similar performance as Zoltar-M. For $C = 1$ and $g = 0.1$, Zoltar-M has zero wasted effort throughout. This occurs because for $N = 500$ the diagnosis only contains the faulty statement (perfect diagnosis), revealing that there is no point in having more than 1 developer debugging the program.

While the above results show to what extent debugging can be efficiently parallelized, in practice information on C is, of course, not available. In the following, we evaluate the added value of multiple-fault diagnosis in estimating the number of debuggers P that can be efficiently deployed in parallel. The plots in Figure 5 show the distribution of the probability (Zoltar-M) or similarity (Zoltar-S, Ochiai, Tarantula) versus the ranking position. For multiple-fault diagnoses each member index is counted as separate position. For cases where the diagnoses are near-perfect ($g = 0.1$) the Zoltar-M distribution clearly exhibits the added information on the program's fault cardinality C (corresponding to the "knee" in the previous plots), whereas the statistical techniques fail to produce any information on C whatsoever (although the Zoltar-S ranking distribution has more dynamics). For high g this relative advantage becomes less as diagnostic quality degrades. Note, that this can be remedied by further increasing N.

V. Empirical Evaluation

Whereas the synthetic observation matrices used in the previous section are populated using a uniform distribution, this is not the case with observation matrices for the behavior of actual programs (different *spectral distribution*). Therefore, in this section we will evaluate the same diagnosis techniques on the Siemens-S set, which was already introduced in Section III-C, and on our multiple-fault extension Siemens-M.

A. Experimental Setup

The Siemens-M set extends the Siemens-S set with program versions that combine several faults from the latter set. The faults can be selectively activated via conditional compilation. In our current experiments, we only use the smallest, and one of largest programs in the Siemens set, `tcas` and `replace`, respectively. The selections of faults that are available in the Siemens-M set are limited by (1) their nature (e.g., a fault in non-executable code, which is not handled by our techniques would effectively reduce a C-fault diagnosis to a $(C-1)$-fault diagnosis), (2) their locations (several faults in Siemens-S have the same statement location), and (3) the number of lines of code involved (we only consider faults that can be attributed to a single line). As a result, Siemens-M supports combinations of eleven faults from Siemens-S for both `tcas` and `replace`.

As mentioned in Section III-C, the `gcov` tool is used to obtain code coverage information for each of the test cases supplied with the programs in the Siemens set. The Zoltar tool set contains utilities for transforming `gcov` output into the line-hit spectra that constitute the first M columns in the observation matrix of Section II-A. The error vector in the last column is constructed by comparing the output of a faulty version of a program with that of the correct version of the program, on a given test case.

For the resulting set of program spectra, Zoltar supports various diagnosis techniques, including Zoltar-M, and the Tarantula, Ochiai, and Zoltar-S coefficients. In the case of Zoltar-M, the presence of duplicate columns, following from the block structure of a program, is exploited in the hitting-set calculation by grouping all identical columns, while maintaining the set of components (lines of code) that they correspond to. This way, larger numbers of components can be handled than in the case of synthetic observation matrices. For M we use the number of lines in the correct versions of `tcas` and `replace` ($M = 174$ and $M = 507$, respectively).

B. Experimental Results

In Figure 6 we show W versus P for both programs when seeded with $C = 1$, $C = 2$, and $C = 3$ faults, respectively. Zoltar-M's expensive hitting set computation was aborted after all diagnosis candidates with cardinality C' had been generated, with $C' = \max(C, 3)$. For `tcas` we also repeated the experiment for $C = 5$ and $C = 10$, but the graphs are similar to those for $C = 2$ and $C = 3$, with the performance of Zoltar-S approaching that of the other methods as C increases. For `replace`, a small number of 4-fault versions were analyzed, but because of the hitting set computation complexity, obtaining results for a significant number of program versions with more than three faults was currently impracticable[2].

The plotted W values are averaged over several different program versions: in case of the plots for $C = 1$, these are all faulty versions in the Siemens-S set that can be attributed to a single line of executable code (30 for `tcas`, and 25 for `replace`). In the case of the $C = 2$ and $C = 3$ plots, these are 40 – 100 randomly selected combinations of faults. The plateau reached by Zoltar-M for `tcas` at high P values is caused by the limited size (ambiguity) of Zoltar-M's diagnosis, removing the need to have them inspected by additional developers.

C. Evaluation

Figure 6(a) and 6(d) confirm the observation of Section IV that for single faults, Zoltar-S is optimal. Although at the end of Section III-C we noted that in a single-fault context, Zoltar-M reduces to Zoltar-S, here Zoltar-M runs in multiple-fault mode, and the presence of cardinality $C' = 3$ diagnoses in the hitting set, as explained above, is the cause for the small differences between these two techniques.

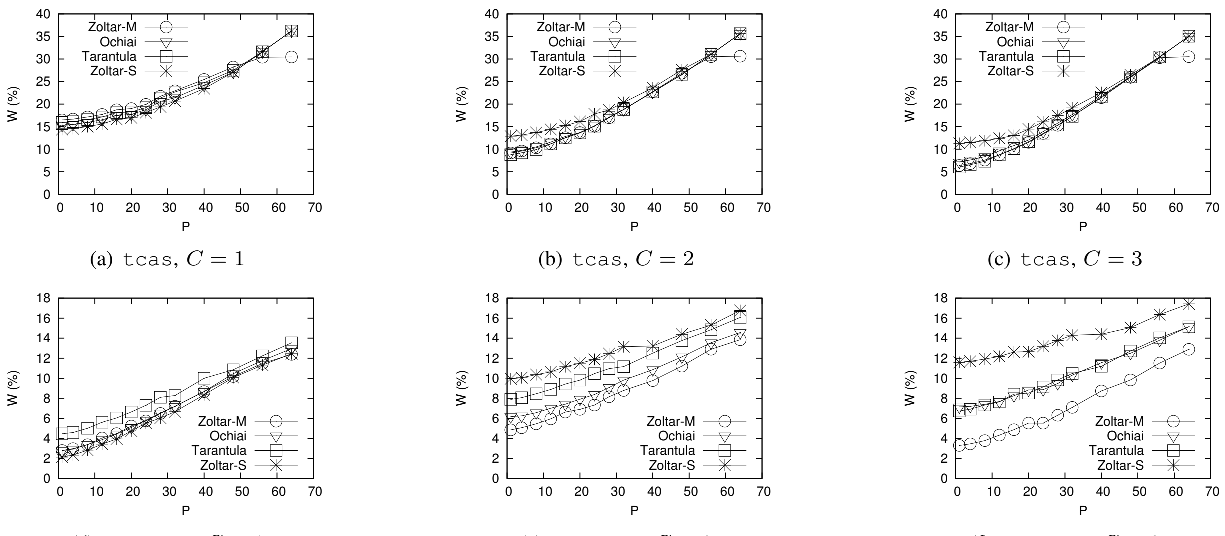
[2]Our latest algorithmic improvement mentioned at the end of Section 3.2 is not yet available for automated experimentation.

Contrary to what we observed in Section IV-B, where Zoltar-S is among the best performing methods for multiple-faults, in Figures 6(b), 6(c), 6(e), and 6(f), Zoltar-S performs worst. This is caused by many non-faulty components that are active in all failed runs. Having $a_{01}(j) = 0$ in Eq. (3), such components fail to be exonerated via the term $\kappa \cdot \frac{a_{01}(j) \cdot a_{10}(j)}{a_{11}(j)}$, and will therefore rank high, leading to a lower quality diagnosis. While in the synthetic observation matrices it is unlikely that a component is active in all failed runs, this is quite common in software (e.g., statements that are always executed).

As shown in Figures 6(b), 6(c), 6(e), and 6(f), for $C = 2$ and $C = 3$, Zoltar-M generally outperforms the statistical techniques, but for `tcas`, its performance is quite close to that of the SFL approaches using the Ochiai and Tarantula coefficients. This can be attributed to the following two related effects. First, the `tcas` faults that are available for making multiple-fault versions have a higher goodness factor ($g = 0.95$) than those available for `replace` ($g = 0.86$), making the diagnosis problems for the multiple-fault `tcas` versions inherently more difficult. The rationale is that for faults whose observation matrix inherently does not permit a good diagnosis (e.g., because the activity of a non-faulty component accidentally coincides with the occurrence of failures), all appropriate techniques will yield an equally bad diagnosis on average. Referring back to the discussion at the end of Section IV-B, the high values for g also explain why on average, no technique achieves optimal diagnostic quality on the Siemens-S and Siemens-M faults, and the consequent absence of a "knee" in the $P - W$ graph of Zoltar-M.

The second effect that contributes to the difference in the plots for `tcas` and `replace` is that the variations in control flow in the former program are extremely limited, while essentially, this is what the diagnosis methods are based on. As an illustration, for the correct version of `tcas`, the observation matrix that follows from the 1,608 test cases that accompany the program contains many duplicate rows and columns: the number of unique rows (spectra) and columns (component behavior profiles) are 8 and 14, respectively. In comparison, the 5,542 test cases of `replace` lead to 2,023 different spectra, and 91 different behavior profiles, providing much more information to base the diagnosis on.

The latter observation confirms our expectation that the effectiveness of automated diagnosis techniques generally improves with program size. As an illustration, near-zero wasted effort is implied by the experiments with SFL on a 0.5 MLOC industrial software product, reported in [20]. In summary, `tcas` is too simple, and Figures 6(e) and 6(f) can be expected to be the more representative of multiple-fault debugging in a realistic development environment. From this we conclude that Zoltar-M can be expected to yield a significant improvement of debugging efficiency over the statistical methods in the multiple-fault case.



(a) tcas, $C = 1$ (b) tcas, $C = 2$ (c) tcas, $C = 3$

(d) replace, $C = 1$ (e) replace, $C = 2$ (f) replace, $C = 3$

Figure 6. Wasted effort for 1 - 64 developers on the Siemens-M benchmark

VI. Related Work

In logic (model-based) reasoning approaches to automatic software debugging, the program model is typically generated using static analysis. In the work of Mayer and Stumptner [13] an overview of the techniques to automatically generate program models is given. They conclude that the models generated by means of abstract interpretation [12] are the most accurate for debugging. Model-based approaches also include the work of Wotawa, Stumptner, and Mayer [18]. Although model-based diagnosis inherently considers multiple-faults, thus far the above software debugging approaches only consider single faults. Apart from this, our approach differs in the fact that we use program spectra as dynamic information on component activity, which allows us to exploit execution behavior, unlike static approaches. Furthermore, our approach does not rely on the approximations required by static techniques (i.e., incompleteness).

Statistical approaches are very attractive from complexity-point of view. Well-known examples are the Tarantula tool [10], the Nearest Neighbor technique [16], the Sober tool [11], and the Ochiai coefficient [2]. Although differing in the way they derive the statistical fault ranking, all techniques are based on measuring program spectra. Examples of other techniques that do not require extra knowledge of the program under analysis are the Delta Debugging technique [19] and the dynamic program slicing technique [7].

Essentially all of the above work has mainly been studied in the context of single faults, except for recent work by Jones, Bowring, and Harrold [9], which is motivated by the obvious advantages of parallel debugging with respect to development time reduction. They use clustering techniques to identify traces (rows in O) which refer to the same fault, after which Tarantula is applied to each cluster of rows. While our work has the same motivation, our approach is based on logic reasoning instead of clustering. Although both introduce an increase of computational complexity, compared to the aforementioned statistical approaches, our hitting set analysis approach is asymptotically optimal, while in the clustering approach there is a possibility that multiple developers will still be effectively fixing the same bug. As their parallel debugging approach has only been evaluated in a restricted empirical context, our results, e.g., for the Siemens programs, cannot yet be compared.

VII. Conclusions and Future Work

In this paper we have presented a multiple-fault localization technique, Zoltar-M, which is based on the dynamic, spectrum-based measurement approach from statistical fault localization methods, combined with a logic (and probabilistic) reasoning approach from model-based diagnosis, inspired by previous work in both separate disciplines [2], [6]. We have compared the performance of Zoltar-M with Tarantula and Ochiai, which are amongst the best known statistical SFL approaches, as well as a new statistical SFL technique, coined Zoltar-S, derived by us as a by-product of our reasoning approach, and shown to be optimal for single-fault programs ($C = 1$).

Our synthetic experiments show that both the reasoning and statistical approaches have the same general properties with respect to the influence of the parameters we introduced, viz, number of components M, number of test cases M, testing code coverage r, testing fault coverage g, and fault cardinality C. For low g both approaches yield near-perfect quality for relatively small N, while for high g (typical for many components in practice) a much larger

N is required for good diagnosis. In most cases it is Zoltar-S that outperforms Zoltar-M, which for $C > 1$ is due to the fact that all components are involved in different runs with the same probability, making it easy for Zoltar-S to pinpoint the faulty ones. Despite these small differences, Zoltar-M's ranking probability distribution clearly provides information on the program's potential debugging parallelism while statistical techniques fail to provide any information.

Our results on two multiple-fault programs of our newly created Siemens-M benchmark suggest that for programs with small spectral distribution variability (and high g value) both approaches do not significantly differ. For the larger program much more test information is available (N), the g parameter is somewhat lower, and the spectral distribution is highly non-uniform. In this case (for $C > 1$) Zoltar-M clearly outperforms all statistical approaches. The disparity with the synthetic results is due to the particular spectral distribution properties of real programs (such as components being executed in all failed runs). Aimed at providing a first-order understanding of the impact of some of the main parameters on diagnostic performance, our simple, probabilistic program model is still far from being able to accurately account for real program behavior.

Although both the reasoning and statistical approach are based on the same (spectral) information, our reasoning approach generally produces improved diagnostic information, in terms of debugging effort and/or (most notably) potential debugging parallelism. Nevertheless our results also indicate that even in the multiple-fault case statistical approaches are by no means outclassed by our reasoning approach, a result that was not initially anticipated. Given the higher complexity of the reasoning approach there may be situations where application of a statistical technique such as Ochiai or Zoltar-S may be preferred over Zoltar-M. In this respect we believe this result may be relevant in the context of the multiple-fault / parallel debugging work by Jones, Bowring, and Harrold [9]. Provided their clustering approach produces spectral partitions that apply to a single fault our results would suggest the use of Zoltar-S, rather than Tarantula.

Despite the higher complexity of Zoltar-M than the statistical approaches, compared to other model-based reasoning approaches, Zoltar-M can handle larger programs (currently up to roughly 500 lines with up to 10 faults) while preliminary experiments have already indicated orders of magnitude of speedup potential. Future work will therefore clearly include algorithmic improvement of our current minimal hitting set algorithm in the statistically-directed search direction mentioned earlier. These improvements pave the way for much more elaborate experimentation

References

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

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

[3] J. de Kleer. Diagnosing intermittent faults. In *Proc. DX'07*.

[4] J. de Kleer, A. K. Mackworth, and R. Reiter. Characterizing diagnoses and systems. *Artificial Intelligence*, 56:197–222, 1992.

[5] J. de Kleer and B. C. Williams. Diagnosing multiple faults. *Artif. Intell.*, 32(1):97–130, 1987.

[6] A. Feldman, G. Provan, and A. J. C. van Gemund. Computing minimal diagnoses by greedy stochastic search. In *Proc. AAAI'08*.

[7] N. Gupta, H. He, X. Zhang, and R. Gupta. Locating faulty code using failure-inducing chops. In *Proc. ASE'05*.

[8] M. J. Harrold, G. Rothermel, R. Wu, and L. Yi. An empirical investigation of program spectra. In *Proc. PASTE'98*.

[9] J. A. Jones, J. F. Bowring, and M. J. Harrold. Debugging in parallel. In *Proc. ISSTA'07*.

[10] J. A. Jones, M. J. Harrold, and J. Stasko. Visualization of test information to assist fault localization. In *Proc. ICSE'02*, pages 467–477, Orlando, Florida, USA, May 2002.

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

[12] W. Mayer and M. Stumptner. Abstract interpretation of programs for model-based debugging. In *Proc. IJCAI'07*.

[13] W. Mayer and M. Stumptner. Models and tradeoffs in model-based debugging. In *Proc. ASE'08*.

[14] J. Pietersma and A. J. C. van Gemund. Temporal versus spatial observability tradeoffs in model-based diagnosis. In *Proc. SMC'06*.

[15] R. Reiter. A theory of diagnosis from first principles. *Artif. Intell.*, 32(1):57–95, April 1987.

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

[17] R. Vayani. Improving automatic software fault localization, July 2007. Master's thesis, Delft University of Technology.

[18] F. Wotawa, M. Stumptner, and W. Mayer. Model-based debugging or how to diagnose programs automatically. In *Proc. IAE/AIE'02*.

[19] A. Zeller. Isolating cause-effect chains from computer programs. In *Proc. FSE'02*.

[20] P. Zoeteweij, R. Abreu, R. Golsteijn, and A. J. C. van Gemund. Diagnosis of embedded software using program spectra. In *Proc. ECBS'07*.