Fault Interference and Coupling Effect

Chunrong Fang, Yang Feng, Qingkai Shi, Zicong Liu, Shuying Li, Baowen Xu*
State Key Laboratory for Novel Software Technology
Nanjing University, Nanjing 210093, China
E-mail: *bwxu@nju.edu.cn

Abstract

Any program may contain more than one fault, and these faults may interfere with each other in a variety of ways. Software behavior may be affected by the interference, resulting in some uncertain results. Such results have negative impact on many software engineering tasks, including regression testing, fault localization, debugging, fault clustering etc. Therefore, understanding the interference becomes an important topic. This paper investigates the fault interference from the perspective of software construction. We introduce the coupling of software construction in order to explain the reasons for fault interference. We observed that different types of coupling may cause three kinds of fault interference and have different probabilities to make the software strike the fault interference traps. We conducted a preliminary experiment on four industrial programs. The results show that our approach gives a good explanation on fault interference.

1 Introduction

Software is in the process of an explosive growth on complexity. Unfortunately, the maintenance process grows more complicated. This brings more difficulties as the number of faults increases and multiple faults usually interfere with each other in real-world software. Due to multiple bugs interfering with each other, some statistical algorithms for single-bug version of software perform poorly in practice[18, 17]. This problem becomes even more difficult as the number of faults in software increases, and real-world software always have multiple faults. Existing statistical algorithms that focus solely on identifying predictors that correlate with program failure perform poorly, especially when there are multiple bugs. To alleviate the problem, failed test cases are segregated into clusters, such that the failed test cases in each cluster can be mapped to the same causative fault in [8, 15]. However, such approach is impractical. Multiple faults may interfere with each other in complex ways, resulting in an inaccurate clustering results. This may lead to an critical and negative impact on some essential software engineering tasks, such as regression testing, fault localization, fault clustering, etc.

However, only a few studies have been conducted to investigate the root cause of faults and their interferences. Debroy and Wong [2] explored the idea of fault interference (FI) by examining the Siemens suite. It was observed that FI is a common phenomenon in software, whereas the reasons why FI exists and how to alleviate FI are not given. Nicholas and James [5] conducted an experiment on some programs of a larger scale, and they got a persuasive result and presented a more thorough explanation on such phenomena.

In this paper, we performed a comprehensive experiment on four industrial C programs. We gain insights in how faults interfere with each other and some potential patterns are mined. This can help developers understand how coupling affects the interaction between multiple faults, which helps them improve their design and implementation. Further, distances between faults are defined to predict patterns that faults interaction will occur and simplify the process to locate faults. Moreover, our work may help software developers in at least three aspects: understanding and improving software tasks mentioned above, obtaining a new perspective to understand software design and software testing, along with learning how to handle and avoid fault interference.

1.1 Motivation

Software coupling is the degree to which each program module relies on each one another[16]. High coupling degree will expose some disadvantages to the software system: needing a ripple effect to changes in other modules, such that it may be hard to reuse and test, lying confusions in the program logic. Absolutely, FI results from coupling, though not all of coupling will generate FIs.

In order to understand the coupling problem, this work examines one aspect of fault behavior: fault interference,

DOI reference number: 10.18293/SEKE2017-077

which has been observed in a variety of research studies. FI is defined as the change in the behavior of one or more faults due to multiple faults on the FI working together. This work provides a deeper investigation of the FI phenomenon. We expect that a thorough understanding of the root causes of FI will enable research areas that address issues with faults to better cope with the challenges that arise from the presence of multiple faults.

1.2 Contribution

In this paper, we make the following contributions:

- We introduce software engineering knowledge to investigate fault interference in multiple fault programs. We define the fault interference problem. As far as we know, this is the first study to introduce engineering perspective on fault interference with multiple faults.
- We introduce software coupling to explain the root causes for different types of fault interference, based on several types of software coupling. No pioneering studies have made such attempts.
- We conducted a comprehensive experiment on industrial subject programs, and the results show that coupling frequency and coupling values contribute to fault interference.

The rest of paper is organized as follows. Section 2 explains the detail of our approach. The experimental design and the result analysis are presented in Section 3. Related work and discussion are presented in Section 4. The conclusion and future work are presented in Section 5.

2 Approach

2.1 Fault Coupling

Some empirical studies [1, 4] show that the designs and implementations with low coupling and high cohesion will lead to more reliable and maintainable products. The industry and academia have been proposed some theories to measure the coupling. Experts have reached an agreement on the coupling type and hold clear definition and threats in software construction of them [11]. We introduce coupling to measure and predicate fault interference. The following five types [] of coupling are observed in our investigation of fault interference. All of them lead to one or more types of fault interference.

We list them in the coupling degree ascending order:

- C1: Data Coupling, communication via scalar parameters;
- C2: Stamp Coupling, dependency induced by the type of structured parameters;
- C3: Control Coupling, parameters are used to control the behavior of a module;
- C4: Common Coupling, communication via shared global data;
- C5: Content Coupling, one module shares and/or changes the definition of another nodule.

2.2 Fault Interference

There exist some studies on fault interference on programs with multiple faults. However, to the best of our knowledge, the formal definition on fault interference is absence. In this section, we propose the definition of fault interference and then category it into three types.

Definition 2.1 (Fault Interference) *Given a test case t, a program P and some faults $f_1, \cdots, f_k$, it is called that $f_1, \cdots, f_k$ interfere on t if*

$$O(P_{f_1+\cdots+f_k}, t) \neq O(P_{f_1}, t) \vee \cdots \vee O(P_{f_k}, t) \qquad (1)$$

$O(P, t)$ is a boolean function of oracle. That is, $O(P, t) = 1$ indicates that P is failed by running t, otherwise, $O(P, t) = 0$. P_{f_i} is a program with one single fault f_i and $P_{f_1+\cdots+f_k}$ is a program with multiple faults $f_1, \cdots, f_k$. Equation (1) implies an inconsistent behaviors between all single versions and a multiple version. To give a deep insight of the fault interference, we introduce the following definition.

Definition 2.2 *Given a program P and some faults $f_1, \cdots, f_k$ interfere on t, (1) the interference is called β-I if $O(P_{f_1+\cdots+f_k}, t) = 1$;(2) the interference is called α-I if $O(P_{f_1+\cdots+f_k}, t) = 0$; (3) For α-I in (2), the interference is called α_1-I if $\forall i \in [1, k]$, $O(P_{f_i}) = 1$, α_1-I if $\exists i \in [1, k]$, $O(P_{f_i}) = 0$.*

For fault interference, as $O(P_{f_1+\cdots+f_k}, t) \neq O(P_{f_1}, t) \vee \cdots \vee O(P_{f_k}, t)$,$O(P_{f_1+\cdots+f_k}, t) = 0$ implies that $O(P_{f_i}, t) = 1$ for some i; $O(P_{f_1+\cdots+f_k}, t) = 1$ implies $O(P_{f_i}, t) = 0$ for all i. α-I indicates a devoid phenomenon of fault interference. β-I indicates an enhancement phenomenon of fault interference. In particular, we further classify α-I into two types: α_1-I if $O(P_{f_i}, t) = 1$ for all i and α_0-I otherwise. That is, α_1-I could be considered as the strong one of α-I.

Note that there is a key difference between our definition and the previous. As we observed in our experiment, the root causes of α_0-I interference and α_1-I interference are generally different. In our opinion, these two phenomena should not be categorized into one type, though the result seems to be similar. In the rest of this paper, we just use α_0, α_1, β to denote the α_0-I, α_1-I, β-I.

2.3 Research Questions

According to the classical software engineering theory, the higher coupling degree is between two blocks, the more possibly exists unpredictable software behavior. To facilitate the discussion, we use $S(F)$ to denote the faults set of the program, and $B(F)$ the block of fault exists. In this case, we can get the mapping: $S(F) \rightarrow B(F)$. $B(F)$ contains the location information, including line number, fault type, existing function head, involving global parameters. $FI_{\alpha\beta}(FI_i, FI_j) \rightarrow B_i, B_j$ we can get construct information between B_i and B_j, which could present the coupling relationship $C(F_i, F_j)$ between F_i and F_j. In this paper, we will investigate research questions as follows.

- RQ1: How frequently the three types of fault interference happen? This question will investigate in what extent fault interference happens and affects program behaviors. Besides, we are interested in whether all the three types have a same change to show up.

- RQ2: Does more fault coupling counts imply more fault interference? This question aims to find the relation between the number of fault coupling and fault interference.

- RQ3: Does high fault coupling value imply more fault interference? Different from $RQ2$, this questions focuses on whether a higher value coupling means a higher possibility of fault interference.

3 Experiment

3.1 Subject Programs

In order to collect sufficient data about fault interference, we used four subject programs: flex, gzip, space and make, all of which are downloaded along with their test sets from the Software-artifact Infrastructure Repository (SIR)[1]. The four programs have been well studied and used as subject programs in many other essential software engineering exploring experiments. Table 1 presents the detailed information. Thanks to the previous researchers [], we excluded some faults that could not cause any failures independently, which results in reducing the number of faults. Besides, using the same approach in [5], we combine faults by randomly selecting a line or block to replace the excluded ones, ensuring that no faulty block is overlapped and the size of fault set is not changed.

[1]http://sir.unl.edu/portal/index.php

Table 1. Subject Programs

Program	Flex	Gzip	Make	Space
Release No.	2.5	1.1.2	3.76.1	2.0
# LOC	14273	7928	35545	6445
# Functions	162	104	268	136
# Faults	20	16	19	33
# Faulty Versions	190	120	171	528
# Test Cases	567	214	1043	13527

3.2 Experiment Setup

To facilitate manual analysis on the causes of fault interference, we simplify the question: two faults interference situation is the best choice, which is of simplicity and representativeness. In this experiment we gathered the pass/fail status of our four subject programs containing one fault independently as well as two faults at same time.

In order to get the pass/fail status, we compare the outputs of test cases run on the fault-seeded versions with that of the golden version. MD5 algorithm is applied to check whether the outputs are the same or not. If the MD5 value is not equal, the execution was marked as a fail; otherwise, it was considered as a pass. With the help of the pass/fail status matrix of all test cases running on all fault-seeded versions, FI types occurrence times, two bugs interacting with each other were recorded by scripts.

The second step is to analyze the root cause of FI. We gather all the fault information, the fault location(in which LOC), fault type, the function header which the fault statements or parameters belong to, the control flow information the faulty block contains, the global parameters which are involved and the local parameters which are used in the block. Furthermore, Egypt[2] is used to generate the function call graph (FCG) and data flow graph (DFG) to assist the software construction analysis procedure. Finally, we apply the information collected to Formula 2, to get the total and average coupling value (CV) of all types of FI.

3.3 Evaluation

For all subject programs are written in C, as described in [1][4], we introduce the Dhama[4] coupling metric that measures the coupling inherent to an individual module M to evaluate the coupling value when FI occurred. The formula is as following:

$$CV(M) = 1 - \frac{1}{i_1 + q_1 i_2 + u_1 + q_2 u_2 + g_1 + q_3 g_2 + w + r}, \quad (2)$$

where q_1, q_2 and q_3 are weight factor to improve the predicator weight. For data and control flow coupling: i_1 is the

[2]http://www.gson.org/egypt/

Table 2. FI Information

	Flex	Gzip	Make	Space
β FI Occurrence Times	0	229	0	138
α_0 FI Occurrence Times	82	103	15	2552
α_1 FI Occurrence Times	652	79	605	13276
Total FI	734	411	620	15966
Single Bug Execution Times	11340	3424	19817	446391
Two Bug Execution Times	107730	25680	178353	7142256
Total Execution times	119070	29104	198170	7588647
FI Occurrence Percent	0.681%	1.60%	0.34%	0.224%
Average of FI in LOC	0.0514	0.0518	0.0174	2.4772
Average of FI in test case	1.2945	1.9205	0.5944	1.1803
Average of FI in function	4.53	3.95	2.31	117.40

Table 3. Coupling Values of FI

FI		Flex	Gzip	Make	Space
β	Total CV	NA	73.0412	NA	39.6661
	FI Occurrence		229		138
	Average CV		0.3189		0.2874
α_0	Total CV	28.6061	5.5943	3.2166	785.849
	FI Occurrence	82	103	15	2552
	Average CV	0.3488	0.2484	0.2144	0.3079
α_1	Total CV	316.4958	26.5607	176.329	5520.1608
	FI Occurrence	652	79	605	13276
	Average CV	0.4845	0.3362	0.2914	0.4158

input data number, i_2 is the control parameter number, u_1 is the output data number, and u_2 is the output control parameter number. For global coupling: g_1 is the number of global parameters used as data, and g_2 is the number of global parameters used for control. For environment coupling: w is the number of other modules called from module M, and r is the number of modules calling module M; The formula has a minimum value of g_1 and g_2 is the number of global variables used for control. In our experiment, to concentrate on the FI occurrence, we just calculate the faulty block coupling value, and q_1, q_2, q_3 are assigned to 2 as a heuristic estimate. To simplify the experiment, we treat the faulty block as the fault module. When the FI occurred in two faulty block, we just treat the involved parameters as the factor of our evaluation formula.

3.4 Experiment Result and Analysis

The analysis of the experiment result includes three parts: FI occurrence statistic, the cause investigation, and the relation between coupling value and FI analysis, to answer the three questions proposed in II-C. The FI statistic information is presented in the Table 2. The coupling percentage analysis is shown in Fig. 1, where α, β denote the FI types depicted in the previous section, and the coupling value statistic is in the Table 3.

Fig. 1 presents the percent of each coupling effect in each type of FI, about the Fig. 1. We must explain that the occurrence of FI may be not caused by only one type of coupling. To facilitate statistical analysis and assure the precision, we record all of the coupling type when FI occurred. We use CP_T(X) to denote the T type(data, stamp, control, common, content) of Coupling Proportion(CP) which lead to X(α_0, α_1, β) type of FI, and the T type of coupling count in X type of FI is denoted as CCO_T(X). The CP_T(X) calculated as follows.

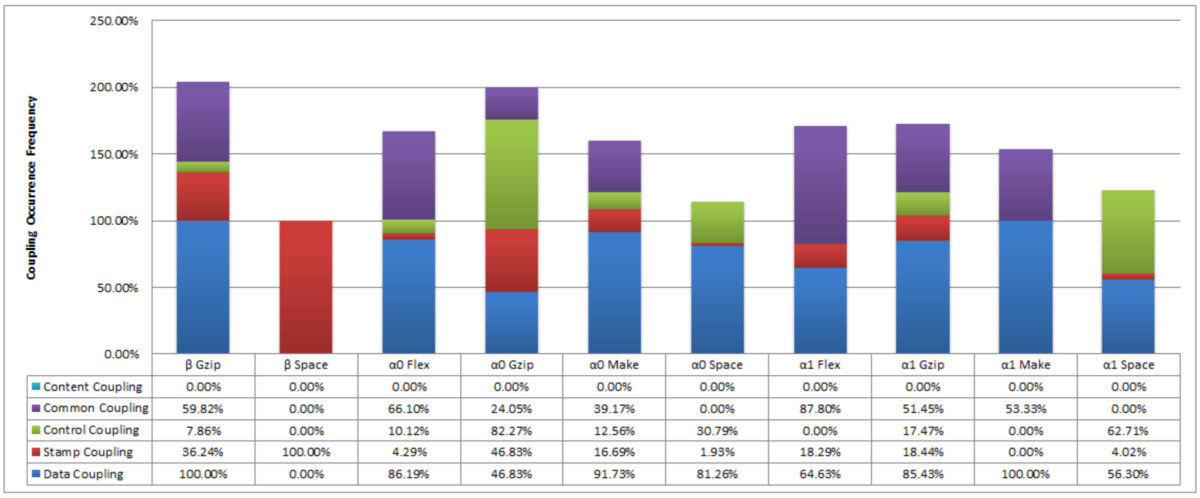
$$CP_T(X) = \frac{CCO_T(X)}{Occurrence\ Number\ of\ X} \tag{3}$$

In this way, the total coupling proportion leads to one type of FI may be over 100 percent.

RQ1: Frequency of fault interference. It is obvious that, the FI occurred in a low frequency, about 1%. In the experiment, we did not observe the FI type, β, in the flex and make, and only a few (about 0.86% = 138/15966) in the Space. The data of Gzip is totally different from other subject programs. The function of Gzip is to compress or decompress the files. We analyze the aberration by tracing the test cases function call routes on the FCG. And the results show that a very high proportion of the test cases called almost all the functions and that each test case has a high code coverage value. In such case, the test case may have a relative high possibility to cover the faulty block. This is considered as the reason why the α_0 and α_1 FI proportion is relatively low in Gzip.

RQ2:The relation between coupling frequency and fault interference. As shown in the Fig. 1, no content coupling is observed in the programs, which may benefit from the well-defined programming style of the four subject programs. However, based on the analysis on the FI types, it reveals that any kind of coupling could lead to FI, that is, a low data coupling frequency corresponds to a low average FI in test cases and a high common coupling frequency generate a high average FI in test cases.

RQ3:The relation between coupling value and fault interference. According to Table 3, all of the subject programs have a higher average CV in α_1 than that in α_0, which indicates that the occurrence of α_1 always results from a much tighter coupling than that resulting in α_0. When we analyze the source code of α FI involved in faulty blocks, we discovered that the main reasons for α_0 and α_1 are totally different: when the α_0 FI occurring, software usually performs correct behavior, which leads the software to skipping the wrong behaviors. In other words, the program seems to behave correct, and jump over another faulty block. In this way the correct behavior cover the wrong behavior and the program seems correct. Another finding is that about 65% α_0 are caused by the use of global parameters. The most important reason leading to α_1 FI is that



	β Gzip	β Space	α0 Flex	α0 Gzip	α0 Make	α0 Space	α1 Flex	α1 Gzip	α1 Make	α1 Space
■ Content Coupling	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
■ Common Coupling	59.82%	0.00%	66.10%	24.05%	39.17%	0.00%	87.80%	51.45%	53.33%	0.00%
■ Control Coupling	7.86%	0.00%	10.12%	82.27%	12.56%	30.79%	0.00%	17.47%	0.00%	62.71%
■ Stamp Coupling	36.24%	100.00%	4.29%	46.83%	16.69%	1.93%	18.29%	18.44%	0.00%	4.02%
■ Data Coupling	100.00%	0.00%	86.19%	46.83%	91.73%	81.26%	64.63%	85.43%	100.00%	56.30%

Figure 1. Coupling Percent Analysis

one of the wrong behavior corrects another wrong behavior by coincidence. And such situation almost occurs in the condition branch. The β FI usually occurs in the situation that: the two independent correct behaviors are not strictly or logically correct, the two faulty blocks form a more fatal fault to some software behavior, and the new fault is easier to be detected for the test case.

In our experiment, we just investigate the FI in two faults as the same with Nicholas and James' excellent research[5]. In general, as the program scale and faults number increase, the FI occurrence frequency will increase. The interference of more than two faults may be a more complicated topic.

4 Related Work and Discussion

Fault localization techniques relying on slicing or statistics usually depend on fault behavior. In [12], James et. al. found that multiple faults can introduce noise, which can decline the effectiveness of fault localization tools. Zheng et al. [18] and Liblit et al. [14] also found a similar phenomenon that multiple faults make feature selection difficult. Denmant et al. [3] coverage-based fault location requires an assumption that multiple faults should be independent with each other. Further, DiGiuseppe et. al. [6] found that fault interference has a large impact on the capability of fault localization techniques on multiple faults program. However, these studies focuses on influence of fault interference on fault localization techniques.

Regression Testing[10] aims to ensure new faults are not introduced when software evolve. In [7], a hierarchy of logic faults are investigated on regression testing with sing fault versions. The fault interference may affect the performance of regression testing.[13] found that a fault may not be detected when it is mixed with other faults.

Fault Interference Debroy and Wong[2] explored the idea of fault interference and they examined whether a test suite perform the same on single-fault versions or multiple-fault versions. Their empirical studies suggest that failure masking is a very quite common phenomenon. DiGiuseppe and Jones [5] provided evidence for the prevalence of fault interference and found that faults obscuring is the most prevalent type. Further, they gave a thorough discussion about the adverse effect of fault interference on many existing studies, such as regression testing and fault localization. Comparing to the existing studies, we introduce software coupling and present explanations on the root causes of fault interference.

5 Conclusion & Future Work

Investigating the fault interference root cause is an important task of software engineering. The interference between faults threats many important software engineering tasks. In this paper, we present a theory to explain the fault interference. The main challenge on this topic is that the developers and testers ignore the fault interference; a natural idea is to propose a theory to explain the phenomenon, predict it and avoid it.

We noticed the phenomenon in our experiment and try to apply the framework to analyze it, but we cannot discuss them in detail due to the limited pages. The approach present in our paper, do not cover how to treat the coupling on the popular Object-Oriented programming, and thanks to the smart previous researchers, there exists some mathematical measuring to measure Object-Oriented, Modular, Aspect-Oriented programming. We will introduce these excellent methodology to investigate the FI.

There is little research on revealing and explaining the fault interference, and the methodology on handling multiple bugs program is in lack. The exploration of multiple bugs is preliminary. There are many aspects about this topic that could be improved or studied in the future. The fault localization algorithm, testing clustering methods, regression testing techniques should take fault interference into consideration. Furthermore, we should build a more detailed theory to explain, predicate and avoid fault interference. For example, predefined specifications with respect faults can be investigated[9]. We will also conduct a more comprehensive experiment to explore the fault interference and take it into consideration to improve other software engineering tasks in the future.

Acknowledgement

This work was partially supported by the National Basic Research Program of China (973 Program 2014CB340702), the National Natural Science Foundation of China (No. 61373013, 61170067), HPI and HPI Research School, Program B for Outstanding PhD Candidate of Nanjing University.

References

[1] J. S. Alghamdi. Measuring software coupling. *Arabian Journal for Science and Engineering*, 33(1):119, 2008.

[2] V. Debroy and W. E. Wong. Insights on fault interference for programs with multiple bugs. In *Software Reliability Engineering, 2009. ISSRE'09. 20th International Symposium on*, pages 165–174. IEEE, 2009.

[3] T. Denmat, M. Ducassé, and O. Ridoux. Data mining and cross-checking of execution traces: a reinterpretation of jones, harrold and stasko test information. In *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*, pages 396–399. ACM, 2005.

[4] H. Dhama. Quantitative models of cohesion and coupling in software. *Journal of Systems and Software*, 29(1):65–74, 1995.

[5] N. DiGiuseppe and J. A. Jones. Fault interaction and its repercussions. In *Software Maintenance (ICSM), 2011 27th IEEE International Conference on*, pages 3–12. IEEE, 2011.

[6] N. DiGiuseppe and J. A. Jones. On the influence of multiple faults on coverage-based fault localization. In *Proceedings of the 2011 international symposium on software testing and analysis*, pages 210–220. ACM, 2011.

[7] C. Fang, Z. Chen, and B. Xu. Comparing logic coverage criteria on test case prioritization. *Science China Information Sciences*, 55(12):2826–2840, 2012.

[8] Y. Feng and Z. Chen. Multi-label software behavior learning. In *Software Engineering (ICSE), 2012 34th International Conference on*, pages 1305–1308. IEEE, 2012.

[9] W. Ghardallou, N. Diallo, and A. Mili. Software evolution by correctness enhancement. In *The 28th International Conference on Software Engineering and Knowledge Engineering, (SEKE)*, pages 605–610. KSI Research Inc., 2016.

[10] T. L. Graves, M. J. Harrold, J.-M. Kim, A. Porter, and G. Rothermel. An empirical study of regression test selection techniques. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 10(2):184–208, 2001.

[11] M. Hitz and B. Montazeri. Measuring coupling and cohesion in object-oriented systems. 1995.

[12] J. A. Jones, M. J. Harrold, and J. Stasko. Visualization of test information to assist fault localization. In *Proceedings of the 24th international conference on Software engineering*, pages 467–477. ACM, 2002.

[13] J.-M. Kim, A. Porter, and G. Rothermel. An empirical study of regression test application frequency. In *Proceedings of the 22nd international conference on Software engineering*, pages 126–135. ACM, 2000.

[14] B. Liblit, M. Naik, A. X. Zheng, A. Aiken, and M. I. Jordan. Scalable statistical bug isolation. *ACM SIGPLAN Notices*, 40(6):15–26, 2005.

[15] C. Liu and J. Han. Failure proximity: a fault localization-based approach. In *Proceedings of the 14th ACM SIGSOFT international symposium on Foundations of software engineering*, pages 46–56. ACM, 2006.

[16] W. P. Stevens, G. J. Myers, and L. L. Constantine. Structured design. *IBM Systems Journal*, 13(2):115–139, 1974.

[17] A. X. Zheng, M. I. Jordan, B. Liblit, and A. Aiken. Statistical debugging of sampled programs. In *NIPS*, volume 16, 2003.

[18] A. X. Zheng, M. I. Jordan, B. Liblit, M. Naik, and A. Aiken. Statistical debugging: simultaneous identification of multiple bugs. In *Proceedings of the 23rd international conference on Machine learning*, pages 1105–1112. ACM, 2006.