# The Impact of Rare Failures on Statistical Fault Localization: the Case of the Defects4J Suite

Yiğit Küçük,
Dept. of Computer and Data Sciences
Case Western Reserve University
Cleveland, Ohio, USA 44106
yigit.kucuk@case.edu

Tim A. D. Henderson,
Google Inc.
1600 Amphitheatre Pkwy
Mountain View, California, USA 94043
tadh@google.com

Andy Podgurski
Dept. of Computer and Data Sciences
Case Western Reserve University
Cleveland, Ohio, USA 44106
podgurski@case.edu

***Abstract*—Statistical Fault Localization (SFL) uses coverage profiles (or "spectra") collected from passing and failing tests, together with statistical metrics, which are typically composed of simple estimators, to identify which elements of a program are most likely to have caused observed failures. Previous SFL research has not thoroughly examined how the effectiveness of SFL metrics is related to the proportion of failures in test suites and related quantities. To address this issue, we studied the Defects4J benchmark suite of programs and test suites and found that if a test suite has very few failures, SFL performs poorly. To better understand this phenomenon, we investigated the precision of some statistical estimators of which SFL metrics are composed, as measured by their coefficients of variation. The precision of an embedded estimator, which depends on the dataset, was found to correlate with the effectiveness of a metric containing it: low precision is associated with poor effectiveness. Boosting precision by adding test cases was found to improve overall SFL effectiveness. We present our findings and discuss their implications for the evaluation and use of SFL metrics.**

## I. Introduction

The Defects4J repository [1] provides automatic fault localization researchers with much of what they require to do meaningful empirical evaluations and comparisons of automated fault localization techniques: a number of substantial "real world" subject programs, each with multiple faulty versions and a test suite that triggers failures of those versions. However, for the purposes of evaluating and comparing statistical fault localization (SFL) techniques [2], which measure statistical associations between program failures and runtime events at individual program locations, Defects4J presents a major problem: for most faulty program versions, the corresponding test suite exhibits *very few failures*. Given that program failures are *rare events*, a statistical test of the hypothesis that covering a given program statement (or other program element) has a *causal effect* on the occurrence of failures will usually be under-powered [3]. Similarly, an estimate of the causal effect of covering a given statement on the occurrence of failures will often be inaccurate — regardless of confounding bias.

Typical "suspiciousness" metrics used in SFL (e.g., to rank statements for examination by programmers) may be viewed as *ad hoc* causal-effect measures, which do not adjust for confounding bias [2]. Even in the absence of confounding, we have found that these metrics do not generally perform well, due to *imprecision* (sampling variability), if any of the following subsets of the test set is too small: the overall number of tests; the number of failing tests; and the number of tests that cover a particular statement. The relative precision of a statistical estimator is characterized by its *coefficient of variation* (CV) [4], [5], which is the ratio $\sigma/\mu$ of its standard deviation (standard error) to its mean. The higher an estimator's CV is, the lower its relative precision. The CV for an estimate of a very small proportion or probability can be large even if $\sigma$ is small.

Many SFL metrics include one or more of the following terms (or minor variations of them) either explicitly or implicitly: (a) the proportion of failing tests among all tests in the test suite; (b) the proportion of tests that cover a given statement $s$ among all failing tests; or (c) the proportion of failing tests among all tests that cover a given statement $s$ ([2], [6], [7]). These terms are in fact statistical estimators, which are affected by the frequency of failures. In this paper we shall demonstrate that when such an estimator is computed for a test suite with very few failures, it is likely to have low relative precision, as measured by its coefficient of variation. We also provide empirical evidence that this results in poor fault localization performance.

## II. Background

### A. Defects4J Framework

One of the concerns fault localization studies often face is related to the evaluation of proposed metrics or techniques. Previous studies have employed a variety of subject programs, faults, and test suites, such as those available from the Software-artifact Infrastructure Repository [8]. Typically, the subject programs used in these studies have been fairly small in size and the faults have been simple, one-line faults, often generated by program mutation [9]. Faults of omission are uncommon in these subject programs.

The Defects4J repository [1], which is composed of six large Java programs, 395 versions with real faults, and their test suites, is a promising alternative. In Defects4J, 76% of the faults span multiple program lines, and 30% of the faults involve code omissions. A complete dissection of the Defects4J dataset can be found in the recent paper by Sobreira *et al.* [10], which describes the types of faults and fixes for each subject program. In the sequel, we will use the

terms "subject program" and "project" interchangeably when discussing Defects4J.

| Identifier | Project Name | # of Faulty Versions |
|---|---|---|
| Chart | JFreeChart | 26 |
| Closure | Closure Compiler | 133 |
| Lang | Apache commons-Lang | 65 |
| Math | Apache commons-Math | 106 |
| Mockito | Mockito | 38 |
| Time | Joda-Time | 27 |

TABLE I
DEFECTS4J SUBJECT PROGRAMS AND NUMBER OF FAULTY VERSIONS FOR EACH PROGRAM

Table I characterizes the contents of the Defects4J repository. The faulty program versions for each project are obtainable via the *checkout* command from the corresponding project repositories. Each faulty version is accessible by its *project ID* and *fault ID*. For each *fault ID* there is a faulty branch and a fixed branch of the same version of the project. They differ only in the line(s) where the fault is fixed. For instance, the first faulty (buggy) version of the project *Chart* is identified as "-p Chart -v 1b" and the fixed branch of the same faulty version is referenced by "- p Chart -v 1f" in the *checkout* command. Fixes made to the faulty versions of a project range in extent from replacement of individual operators to insertion or removal of chunks of code in different locations in the project [10]. All the changes are well documented and are accessible with the *info* command. All the faults are arranged in chronological order; for instance, in project *Chart*, faulty version 1 is the most recent version in the repository and faulty version 26 is the earliest version.

In the test suite for each faulty Defects4J program version, there exists a failing test case, which is called a *relevant* test, that reveals the fault in that version. These failures are not random, nor do they rely on the execution order of the tests. By default, the tests relevant to a certain fault are executed using the *test* command. However, the framework allows tests to be run manually either as a single test case or as a group of tests. Defects4j can report properties of a project version such as the list of relevant test cases, the tests that trigger a fault, and the classes loaded for the tests. The aforementioned features of Defects4J allow parts of the test suite to be analyzed independently and provide control of how the tests are run for a given program version.

### B. Common Elements of Coverage-Based SFL Metrics

Statistical fault localization techniques often analyze code-coverage profiles (or "spectra") in order to correlate coverage of individual program elements (such as statements, basic blocks, or functions) with program failures. This is done using measures of statistical association, which are known as *coverage-based SFL (or SBFL) metrics* or *suspiciousness metrics*. Different metrics can be defined using some common notation, which is summarized in Table II.

Various studies have pointed out commonalities between SFL metrics (e.g., [2], [6], [7]). In addition to the quantities listed in Table II, certain subexpressions occur in a number of metrics, either explicitly or implicitly. Three examples of such subexpressions are [7]: (a) the proportion of failing tests among all tests in the test suite, $\frac{n_f}{n}$, which is an estimator for the probability $\Pr[failure]$; (b) the proportion of tests that cover a given statement $s$ among all failing tests, $\frac{n_f(s)}{n_f}$, which is an estimator of the conditional probability $\Pr[s\ covered|failure]$; or (c) the proportion of failing tests among all tests that cover a given statement $s$, $\frac{n_f(s)}{n(s)}$, which is an estimator of $\Pr[failure|s\ covered]$.[1] For example, the Ochiai metric [11] can be written as

$$\sqrt{\frac{n_f(s)}{n_f} \cdot \frac{n_f(s)}{n(s)}} \approx \sqrt{\Pr[s|failure] \Pr[failure|s]}$$

| Notation | Definition |
|---|---|
| $n$ | Total number of test cases in the test suite |
| $n(s)$ | Number of test cases that execute statement $s$ |
| $n_p$ | Number of passing test cases |
| $n_f$ | Number of failing test cases |
| $n_p(s)$ | Number of tests that execute statemment $s$ and pass |
| $n_f(s)$ | Number of tests that execute statement $s$ and fail |

TABLE II
NOTATION USED IN DEFINING ASSOCIATION MEASURES

| Project | All Tests | Passing | Failing |
|---|---|---|---|
| Chart | 185 | 181 | 4 |
| Closure | 3351 | 3348 | 3 |
| Mockito | 671 | 668 | 3 |
| Time | 2525 | 2522 | 3 |
| Lang | 94 | 92 | 2 |
| Math | 172 | 170 | 2 |

TABLE III
AVERAGE NUMBERS OF TESTS PER DEFECTS4J SUBJECT PROGRAM

## III. CHARACTERIZING THE PRECISION OF ESTIMATES

The precision of a statistical estimator is often characterized by its variance or standard deviation (standard error) [4], and an estimator with high variance is considered imprecise. However, when estimating the frequency or probability of rare events, it is preferable to use the coefficient of variation of the estimator, $\sigma/\mu$, where $\sigma$ is the standard deviation of the estimator and $\mu$ is its expected value [5]. The CV characterizes the *relative precision* of an estimator. For an unbiased estimator, whose expected value is equal to the estimand (the parameter to be estimated), a high CV value indicates that the estimator is highly variable relative to the magnitude of the estimand. Dixon *et al.* [5] point out, in the context of ecological studies, that when the precision of an estimate is low, it is more difficult to detect differences between groups of interest or patterns in the frequency of rare events. In SFL, low precision can cause correct program statements to be ranked higher than faulty statements for examination by programmers. Dixon *et al.* also state that if an event is truly rare (probability $\pi \leq 0.01$) then its maximum likelihood estimate $\hat{p} = n/N$, where $n$ is the number of events observed and $N$ is the total number of observations, has "reasonable" precision ($CV(\hat{p}) \leq 10\%$) only when $N > 1000$,

[1] Each of these estimators is a sample proportion and so is unbiased in the sense that its expected value is equal to the probability to be estimated. However, their use does not prevent confounding bias when estimating causal effects, which are defined in terms of counterfactuals [2].

whereas for $N < 100$, $CV(\hat{p})$ may exceed 300%. (Note that the standard error of $\hat{p}$ is $\sqrt{\hat{p}(1-\hat{p})/N}$.) Of course, using a larger sample size generally entails higher costs. In fault localization, these include the costs of constructing new test cases, executing them, and evaluating the results.

| Project | Failure Prop. | CV(a) (%) | CV(b) (%) | CV(c) (%) |
|---|---|---|---|---|
| Chart | 0.0216 | 49.4 | 65.2 | 55.0 |
| Closure | 0.0009 | 57.7 | 83.5 | 83.3 |
| Mockito | 0.0045 | 57.6 | 77.5 | 76.5 |
| Time | 0.0012 | 57.7 | 71.1 | 70.2 |
| Lang | 0.0213 | 69.9 | 88.1 | 80.2 |
| Math | 0.0116 | 70.2 | 91.5 | 86.6 |

TABLE IV
AVERAGE FAILURE PROPORTION AND COEFFICIENTS OF VARIATION OF (a) $\frac{n_f}{n}$, (b) $\frac{n_f(s)}{n_f}$, AND (c) $\frac{n_f(s)}{n(s)}$, FOR EACH DEFECTS4J SUBJECT PROGRAM

Earlier we indicated that the proportions of failing tests in the test suites for many faulty Defects4J subject program versions are very low. Table III presents the average numbers of tests overall, of passing tests, and of failing tests, over the faulty versions of each subject program, each of which has its own test suite. Table IV presents the average failure proportion and the average coefficients of variation, over all statements in all faulty versions of each Defects4J subject program, of each of the subexpressions mentioned in Section II-B as occurring in a number of SFL metrics. That is, Table IV presents the average failure proportion and the average values of $CV(\frac{n_f}{n})$, $CV(\frac{n_f(s)}{n_f})$, and $CV(\frac{n_f(s)}{n(s)})$. It is clear that each average failure proportion is low and each CV is high, with the CVs ranging from 49.4 to 91.5. These values indicate that the relative precision of the three common subexpressions is low for the Defects4J faulty versions, and it suggests that any SFL metric that contains one or more of the subexpressions, such as the Ochiai metric, is at risk of being imprecise when the failure proportion is low (unless it downweights those terms).

## IV. PRECISION VERSUS FAULT LOCALIZATION EFFECTIVENESS

We now investigate the relationship between estimation precision and fault localization effectiveness using the Defects4J suite.

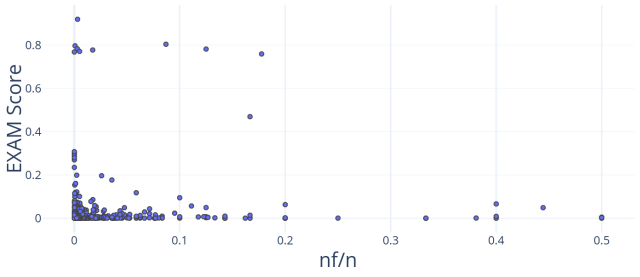


Fig. 1. Scatter plot of $\frac{n_f}{n}$ versus *EXAM* score achieved with the Ochiai metric, for all faulty Defects4J program versions

Specifically, we relate the values of the common SFL-metric subexpressions $\frac{n_f}{n}$, $\frac{n_f(s)}{n_f}$ and $\frac{n_f(s)}{n(s)}$, as well as their coefficients of variation, to the effectiveness of the Ochiai metric as measured by its *EXAM* score [12], for all 395 faulty program versions in Defects4J. Recall that the Ochiai metric includes the latter two proportions as subexpressions. The *EXAM* score is the percentage of program statements that a developer must examine, in non-increasing order of their suspiciousness scores, to find the first faulty statement [12]. (It was assumed that if a faulty location $L$ has the same suspiciousness score as other locations, half of the tied locations must be examined to find $L$ on average.) For calculating both the *EXAM* score and the CV values, we modified scripts provided by Pearson *et al.* [13], which use *Gzoltar* [14] for collecting coverage information.

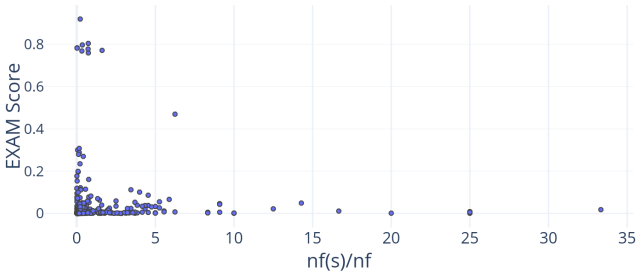


Fig. 2. Scatter plot of average value of $\frac{n_f(s)}{n_f}$ versus *EXAM* score achieved with the Ochiai metric, for all faulty Defects4J program versions. The proportion was averaged over all statements in a version.

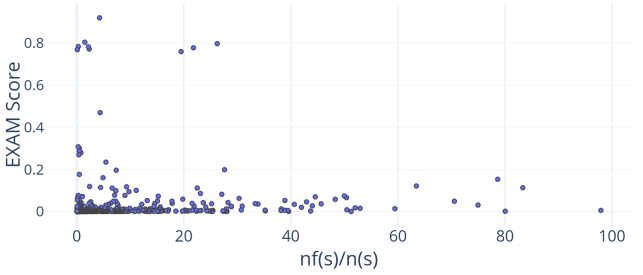


Fig. 3. Scatter plot of average value of $\frac{n_f(s)}{n(s)}$ versus *EXAM* score achieved with the Ochiai metric, for all faulty Defects4J program versions. The proportion was averaged over all statements in a version.

Figure 1 is a scatter plot of the failure proportion $n_f/n$ versus the *EXAM* score achieved with the Ochiai metric, for all 395 faulty Defects4J program versions. Figures 2 and 3 are scatter plots of average values of the proportions $\frac{n_f(s)}{n_f}$ and $\frac{n_f(s)}{n(s)}$, respectively, versus the *EXAM* score achieved with the Ochiai metric, for all faulty Defects4J program versions. The proportions were averaged over all statements in a version. It is evident that low values of the three proportions are associated with high *EXAM* scores, that is, with high fault localization cost.

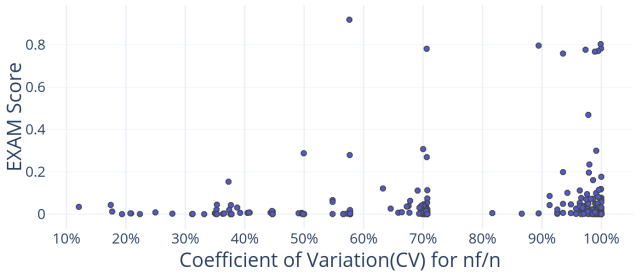


Fig. 4. Scatter plot of $CV(\frac{n_f}{n})$ versus *EXAM* score achieved with the Ochiai metric, for all faulty Defects4J program versions.

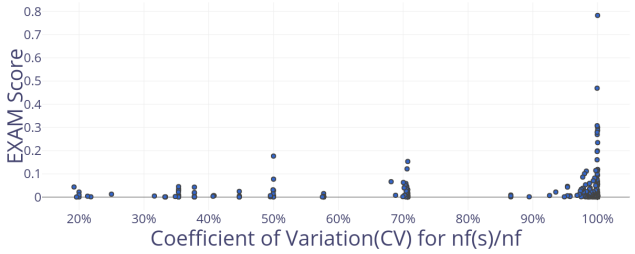


Fig. 5. Scatter plot of $CV(\frac{n_f(s)}{n_f})$ for faulty statement $s$ versus *EXAM* score achieved with Ochiai metric, for all faulty Defects4J program versions.

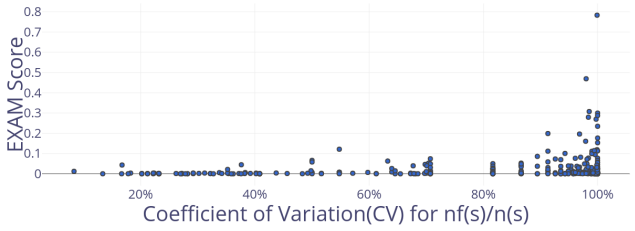


Fig. 6. Scatter plot of $CV(\frac{n_f(s)}{n(s)})$ for faulty statement $s$ versus *EXAM* score achieved with Ochiai metric, for all faulty Defects4J program versions.

Figure 4 shows a scatter plot of $CV(\frac{n_f}{n})$ versus *EXAM* score achieved with the Ochiai metric, for all the faulty Defects4j program versions. Figures 5 and 6 show scatter plots of $CV(\frac{n_f(s)}{n_f})$ and $CV(\frac{n_f(s)}{n(s)})$, respectively, for a faulty statement $s$ versus the *EXAM* score achieved with the Ochiai metric, for all faulty Defects4J program versions. (For a version with multiple faulty statements, only the CV for the lowest ranked statement is plotted.) It is evident that, in both figures, high CV values are associated with high *EXAM* scores, that is, with high fault localization cost, although some some program versions had low *EXAM* scores despite having high CV values. This appears to be due to the fact that the Defects4J test suites each contain a relevant test, which reveals the fault in the corresponding program version. Of courses, in general it is not algorithmically decidable if a given test suite contains a relevant test.

## V. Increasing Sample Size

As mentioned earlier, one basic way of increasing the precision of estimates of rare event frequencies is to increase the sample size. We mentioned that when the Defects4j *test* command is invoked out, only the relevant test for a particular fault is executed. The whole test suite is usually not executed due to performance concerns. Fortunately, it is possible to combine Defects4J test suites for certain versions of a subject program by combining the list of tests to be executed to obtain larger test suites without manually introducing more code to the programs. We exploited this fact to explore the effect on SFL performance (cost) of increasing the sample size.

Specifically, we took the union of test suites (relevant tests) for a number of pairs of faulty program versions, in increasing order of faulty program version numbers. For example, faulty version $V_n$ is combined with $V_{n+1}$, and $V_{n+2}$ is combined with $V_{n+3}$. The two program versions in each pair had different fault locations and different test suites. The faulty program versions were not themselves merged; only the test suites (sets of relevant tests) were merged. This process produced a number of new test suites that were larger than the ones they were composed of, with minimal differences in the program files and code. Table V shows the number of enlarged test suites obtained. We then compared the *EXAM* scores for the Ochiai metric and for the original and modified test suites to see if increasing the sample size improved fault localization effectiveness.

| Project | # of New Test Suites |
|---|---|
| Chart | 10 |
| Time | 10 |
| Lang | 10 |
| Math | 10 |
| Total | 40 |

TABLE V
Number of Enlarged Test Suites Obtained Per Program

Unfortunately, when trying to combine the test suites of *Mockito* and *Closure*, we encountered an error with Gzoltar that prevented us from obtaining enlarged test suites for those programs. The results are displayed in Figure 7.

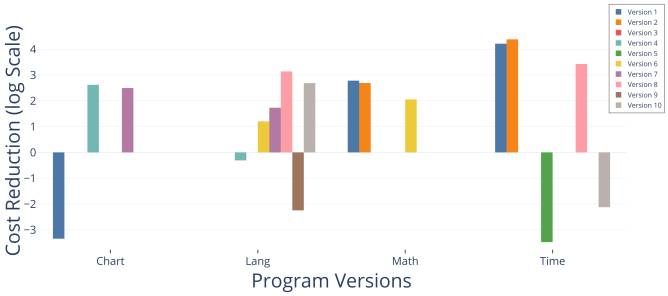


Fig. 7. *EXAM* score difference (cost reduction) for all enlarged test suites, higher is better in this chart.

Out of 40 enlarged test suites, 35 had *EXAM* scores that were no higher, and that in several cases were lower, than

for the original suites. This indicates that increasing the test suite size reduced fault localization cost in these cases. The remaining 5 suites had higher *EXAM* scores than original suites. We believe that in the latter cases the added tests did not increase coverage of the faulty statements.

## VI. Conclusion

We have examined how the precision of a statistical fault localization metric affects its performance, and how a test suite that induces few failures can cause SFL metrics to exhibit low relative precision, as measured by their coefficient of variation, and poor performance. We have also provided empirical evidence that this occurs with the Defects4J benchmark suite, and we examined the effect on precision of increasing the sample size. Our results suggest that evaluating any SFL metric using fault localization benchmarks with few failure-inducing tests is very likely to ensure that the metric exhibits poor performance. One important implication of this is that SFL metrics are applicable only in scenarios in which significant numbers of failure-inducing inputs, as well as success-inducing inputs, can be obtained and labeled. The such scenarios would occur, for example, when many end-users report failures but profiles/spectra are collected automatically.

Our results also indicate that when deciding on the applicability of statistical fault localization to a given program and test suite, it is prudent to compute the coefficients of variation (CV) of SFL metrics under consideration or their subexpressions. If this indicates that a metric is likely to exhibit low precision, that implies it is not the right choice for this application. Adding new test cases that increase coverage and test case diversity more generally may improve SFL accuracy.

## VII. Future Work

We intend to replicate our study with additional collections of subject programs and with additional SFL metrics. In particular, Defects4J contains only single-fault versions of programs, so a study involving programs with multiple real faults is needed. We also intend to investigate the effects on fault localization performance of removing failing tests from test suites and, separately, of adding relevant tests.

## Acknowledgement

This work was partially supported by NSF award CCF-1525178 to Case Western Reserve University.

## References

[1] R. Just, D. Jalali, and M. D. Ernst, "Defects4j: A database of existing faults to enable controlled testing studies for java programs," in *Proceedings of the 2014 International Symposium on Software Testing and Analysis*. ACM, 2014, pp. 437–440.

[2] G. K. Baah, A. Podgurski, and M. J. Harrold, "Causal inference for statistical fault localization," in *Proceedings of the 19th international symposium on Software testing and analysis*. ACM, 2010, pp. 73–84.

[3] L. Balzer, J. Ahern, S. Galea, and M. van der Laan, "Estimating effects with rare outcomes and high dimensional covariates: knowledge is power," *Epidemiologic methods*, vol. 5, no. 1, pp. 1–18, 2016.

[4] C.-E. Särndal, B. Swensson, and J. Wretman, *Model assisted survey sampling*. Springer Science & Business Media, 2003.

[5] P. M. Dixon, A. M. Ellison, and N. J. Gotelli, "Improving the precision of estimates of the frequency of rare events," *Ecology*, vol. 86, no. 5, pp. 1114–1123, 2005.

[6] L. Lucia, D. Lo, L. Jiang, F. Thung, and A. Budi, "Extended comprehensive study of association measures for fault localization," *Journal of software: Evolution and Process*, vol. 26, no. 2, pp. 172–219, 2014.

[7] S.-F. Sun and A. Podgurski, "Properties of effective metrics for coverage-based statistical fault localization," in *2016 IEEE International Conference on Software Testing, Verification and Validation (ICST)*. IEEE, 2016, pp. 124–134.

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

[9] T. A. Budd, R. A. DeMillo, R. J. Lipton, and F. G. Sayward, "The design of a prototype mutation system for program testing," in *Proceedings of the AFIPS National Computer Conference*, vol. 74, 1978, pp. 623–627.

[10] V. Sobreira, T. Durieux, F. Madeiral, M. Monperrus, and M. de Almeida Maia, "Dissection of a bug dataset: Anatomy of 395 patches from defects4j," in *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2018, pp. 130–140.

[11] R. Abreu, P. Zoeteweij, and A. J. Van Gemund, "An evaluation of similarity coefficients for software fault localization," in *2006 12th Pacific Rim International Symposium on Dependable Computing (PRDC'06)*. IEEE, 2006, pp. 39–46.

[12] E. Wong, T. Wei, Y. Qi, and L. Zhao, "A crosstab-based statistical method for effective fault localization," in *2008 1st International Conference on Software Testing, Verification, and Validation*. IEEE, 2008, pp. 42–51.

[13] S. Pearson, J. Campos, R. Just, G. Fraser, R. Abreu, M. D. Ernst, D. Pang, and B. Keller, "Evaluating and improving fault localization," in *Proceedings of the 39th International Conference on Software Engineering*. IEEE Press, 2017, pp. 609–620.

[14] J. Campos, A. Riboira, A. Perez, and R. Abreu, "Gzoltar: an eclipse plug-in for testing and debugging," in *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*. ACM, 2012, pp. 378–381.