Are Mutation Scores Correlated with Real Fault Detection?

A Large Scale Empirical study on the Relationship Between Mutants and Real Faults

Mike Papadakis
University of Luxembourg
michail.papadakis@uni.lu

Donghwan Shin
Korea Advanced Institute of Science and Technology
donghwan@se.kaist.ac.kr

Shin Yoo
Korea Advanced Institute of Science and Technology
shin.yoo@kaist.ac.kr

Doo-Hwan Bae
Korea Advanced Institute of Science and Technology
bae@se.kaist.ac.kr

ABSTRACT

Empirical validation of software testing studies is increasingly relying on mutants. This practice is motivated by the strong correlation between mutant scores and real fault detection that is reported in the literature. In contrast, our study shows that correlations are the results of the confounding effects of the test suite size. In particular, we investigate the relation between two independent variables, mutation score and test suite size, with one dependent variable the detection of (real) faults. We use two data sets, CoreBench and Defects4J, with large C and Java programs and real faults and provide evidence that all correlations between mutation scores and real fault detection are weak when controlling for test suite size. We also find that both independent variables significantly influence the dependent one, with significantly better fits, but overall with relative low prediction power. By measuring the fault detection capability of the top ranked, according to mutation score, test suites (opposed to randomly selected test suites of the same size), we find that achieving higher mutation scores improves significantly the fault detection. Taken together, our data suggest that mutants provide good guidance for improving the fault detection of test suites, but their correlation with fault detection are weak.

CCS CONCEPTS

• **Software and its engineering → Software testing and debugging**;

KEYWORDS

mutation testing, real faults, test suite effectiveness

ACM Reference format:
Mike Papadakis, Donghwan Shin, Shin Yoo, and Doo-Hwan Bae. 2018. Are Mutation Scores Correlated with Real Fault Detection?. In *Proceedings of ICSE '18: 40th International Conference on Software Engineering , Gothenburg, Sweden, May 27-June 3, 2018 (ICSE '18),* 12 pages.
https://doi.org/10.1145/3180155.3180183

ICSE '18, May 27-June 3, 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5638-1/18/05…$15.00
https://doi.org/10.1145/3180155.3180183

1 INTRODUCTION

What is the relation between mutants and real faults? To date, this fundamental question remains open and, to large extent, unknown if not controversial. Though, a large body (approximately 19% [34]) of the software testing studies rely on mutants.

Recent research investigated certain aspects of the fault and mutant relation, such as the correlation between mutant kills with real fault detection [3, 24] and the fault detection capabilities of mutation testing [8]. Just *et al.* [24] report that there is "a statistically significant correlation between mutant detection and real fault detection, independently of code coverage", while Chekam *et al.* [8] that "fault revelation starts to increase significantly only once relatively high levels of coverage are attained".

Although these studies provide evidence supporting the use of mutants in empirical studies, this is contradictory to the findings of other studies, e.g., study of Namin and Kakarla [28], and to some extent between themselves (as they do not agree on the strength and nature of the investigated relations). Furthermore, there are many aspects of the mutant-fault relation that still remain unknown.

For instance, the study of Just *et al.* [24] did not control for the size of the test suites, which is a strong confounding factor in software testing experiments [21, 27]. This is because a larger test suite is more likely to detect more faults than a smaller one, simply because it contains more tests. At the same time, a larger test suite kills more mutants than a smaller one. Therefore, as both mutation score and test suite size are factors with potential impact on fault detection, it is unclear what is the relation between mutation score and real fault detection, independently of test suite size.

Additionally, the study of Just *et al.* [24] measured the *correlation* between mutant kills and fault detection on Java programs while the study of Chekam *et al.* [8] measured the *actual fault detection* of mutation-based test suites on C programs. Therefore, it is unclear whether the findings on Java programs hold on the C ones (and vice versa) and more generally, whether there is any practical difference between the two evaluation measurements, i.e., correlation analysis between mutant kills and fault detection and actual fault detection rate of mutation-based test suites.

The differences between these two evaluation metrics is important as they are extensively used in empirical studies [36]. Yet, it is unclear whether there are any practically significant differences between them. In case the differences are significant, one could draw different conclusions by using one metric over the other. Thus, investigating the potential differences between these metrics can be useful to other studies that compare test criteria and test methods.

In our study, we use a large number of real faults from real-world C and Java programs. To perform our analysis in a reliable and as generic as possible way, we use the developer test suites, augmented by state-of-the-art test generation tools, KLEE for C [7], Randoop [32] and EvoSuite [15] for Java. These tools helped us composing a large, diverse and relatively strong test pool from which we sample multiple test suites. To ensure the validity of our analysis, we also repeat it with the developer and automatically generated test suites and found insignificant differences.

One key result is that the correlation between mutant kills and real fault detection drops significantly and became weak when controlling for test suite size. We also perform regression analysis and show that both size and mutation score independently influence the fault detection. Both of them, alone achieve a good level of correlation, which becomes weak when the other is put under experimental control. Overall, the combination of both size and mutation score achieves statistically significantly better fits than the size and mutation score alone.

Interestingly, a deeper analysis reveals that mutants are indeed capable of representing the behaviour of real faults. However, these mutants are very few (less than 1% of the involved mutants). This means that only a few mutants play the key role in representing the behaviour of real faults, and thus, mutation scores are subject to 'noise effects' caused by the large numbers of mutants that are, in some sense, "irrelevant" to the studied faults.

Finally, our study investigates the test effectiveness question, i.e., how effective mutants are at finding faults. Correlations measure the extent of the variability of the independent variable, i.e., mutation score, is explained by the variability of the dependent one, i.e., fault detection. However, this does not necessarily imply that a test criterion provides (or not) enough guidance on uncovering faults when fulfilling its test requirements as the relation might be non-linear [8, 12]. We, thus, demonstrate that correlations do not reflect the fault detection capabilities of mutation testing, which can be significant despite the low correlations.

2 MUTATION ANALYSIS

Mutation analysis introduces defects, called *mutants*, which form the test objectives. A test case detects a mutant-defect, when it makes its observable program output different from that of the original program. A detected mutant is call *killed*, while a non-detected one is called *live*.

The ratio of the killed to the total number of mutants is called *mutation score* and represents the degree of adequacy achievement of the test suites. Unfortunately, some mutants cannot be killed as they are functionally equivalent to the original program [35, 39]. These mutants are called *equivalent* and need to be removed from the calculation of the mutation score. However, their identification is done manually as it is an instance of an undecidable problem [2].

Mutation analysis has many applications [31, 36], but the majority of the existing work is using it to support experimentation [34, 36] and the testing process [2, 16, 36]. In the former case, the key question is whether mutants provide results that are representative of those that one could obtain by using real faults. In the latter case, the key question is whether, by killing more mutants one can significantly increase the capability to detect faults.

2.1 Mutants and Real Faults

Studies investigating the relationship between mutants and real faults are summarised in Table 1. Note that Table 1 is strictly restricted to the findings related to the relationship between mutants and real faults. Additional details related to the subject can be found in the recent mutation testing survey of Papadakis *et al.* [36]. As can be seen in the Table, six of the studies concluded that there is a strong connection between mutation score and fault detection, independent of the test suite size. However, studies considering the influence of size report mixed results. Among four such studies, one reports weak correlation, study of Namin and Kakarla [28], one reports some form of strong correlation as it reports minor differences between the mutant and fault detection ratios, study of Andrews *et al.* [3], and the remaining two report that the improvement on fault detection when reaching higher mutation score levels was non-linear, studies of Frankl *et al.* [14] and Chekam *et al.* [8]. Therefore, the emerging question is the one about the relation between size, mutation score, and fault detection.

Furthermore, all these studies have been assessed by employing two main evaluation metrics. These are either some form of correlation analysis between mutant kills and fault detection ratios (studies with references [3, 4, 10, 24, 28]) or the fault detection rate at the higher mutation score levels (studies [8, 14, 37, 38, 40]). Therefore, it is unclear what the relationship between these two evaluation metrics is, and what the relationship implies for the actual application of mutation testing.

Perhaps the first study that investigated the use of mutants as replacements of real faults was that of Daran and Thévenod-Fosse [10]. This study considered a C program of approximately 1,000 lines of code with 12 faults and showed that mutants infect the internal program states in a way that is similar to the way that real faults corrupt program states. In particular, the study reports that 85% of the corrupted states, produced by mutants, were the same with those produced by real faults. Only the 7% were different at the fault introduction state and 8% were different during error propagation. Overall, all the failures caused by real faults were reproduced by the mutants.

Andrews *et al.* [3, 4] used a C program (named space) of approximately 5,000 lines of code with 38 faults and demonstrated that mutant kills and fault detection ratios have similar trends. In a later study, Namin and Kakarla [28] used the same program and fault set and came to the conclusion that there is a weak correlation between mutants and fault detection ratios. Recently, Just *et al.* [24] used a large number of real faults from five Java projects and demonstrated that mutant detection rates have a strong positive correlation with fault detection rates. Since the study of Just *et al.* [24] did not consider test suite size and its results contradict the ones of Namin and Kakarla [28], it remains unclear whether mutation score actually correlates with fault detection when test suite size is controlled.

Papadakis and Malevris [37] used the space program, C program of approximately 5,000 lines of code, with 38 faults and found that mutants provide good guidance towards improving test suites independent of test suite size. Shin *et al.* [40] and Ramler *et al.* [38] came to similar conclusions (mutants can help improving the fault detection of test suites). However, both these studies did not account for the size effects of the test suites.

Table 1: Summary of studies investigating the relationship between mutants and real faults.

Author(s) [Reference]	Year	Largest Subject	Language	Considered Test Size	No Faults	Summary of Scientific Findings
Daran & Thévenod-Fosse [10]	96	1,000	C	×	12	Mutants result in failures and program data states that are **similar** to those produced by real faults.
Frankl *et al.* [14]	'97	78	Fortran, Pascal	✓	9	**Fault detection** probability is increasing at higher mutation score levels. The increase is non-linear.
Andrews *et al.* [3]	'05	5,000	C	✓	38	Mutants detection ratios are **representative** of fault detection ratios
Andrews *et al.* [4]	'06	5,000	C	×	38	Mutants detection ratios are **representative** of fault detection ratios
Papadakis & Malevris [37]	'10	5,000	C	×	38	1^{st} order mutation has higher **fault detection** than 2^{nd} order and mutant sampling. There are significantly less equivalent 2nd order mutants than 1^{st} order ones.
Namin & Kakarla [28]	'11	5,000	C	✓	38	There is a weak **correlation** between mutant detection ratios and real fault detection ratios
Just *et al.* [24]	'14	96,000	Java	×	357	There is a strong **correlation** between mutant detection ratios and real fault detection ratios
Shin *et al.* [40]	'17	96,000	Java	×	352	Distinguishing mutation adequacy criterion has higher **fault detection** probability than strong mutation adequacy criterion
Ramler *et al.* [38]	'17	60,000	Java	×	2	Mutation testing helps improving the test suites of a safety-critical industrial software system by increasing their **fault detection** potential.
Chekam *et al.* [8]	'17	83,100	C	✓	61	Mutation testing provides valuable guidance for improving test suites and revealing real faults. There is a strong connection between mutation score increase and **fault detection** at higher score levels.
This paper	'18	96,000	C & Java	✓	420	There is a weak **correlation** between mutation score and real fault detection. Despite the weak correlations, fault detection is significantly improved at the highest score levels.

Finally, Frankl *et al.* [14], experimented with some small method-units (in Pascal and Fortran) and found that mutants provide good guidance towards improving test suites, even when test suite size is controlled. Similarly, Chekam *et al.* [8] used real faults from four real-world C projects and demonstrated that there is a strong connection between mutation score attainment and fault detection only at higher score levels.

Overall, despite the results found in the literature, our understanding of the relationship between the test suite size, mutation score, and real fault detection remains limited as none of the studies investigates them with a large set of real faults and real-world programs. Moreover, no previous study investigates whether there are practical differences between the correlation analysis and the fault detection rate of test suites when used as evaluation metrics of software testing experiments.

2.2 Mutants and Hand-Seeded Faults

Due to lack of real faults, many studies used hand-seeded faults to simulate test effectiveness (i.e., fault detection capability).

Wong and Mathur [43] demonstrate that mutation testing has higher fault detection potential than data-flow. Offutt *et al.* [30] also found that mutation is more effective than data-flow (mutation detected on average 16% more faults than the data flow) but at a higher cost (measured as the number of test cases).

Li *et al.* [26] experimented with coverage criteria and compared them with mutation testing, in terms of the number of faults detected. Their results showed that mutation testing detected more faults than the coverage criteria. Interestingly, the same study reports that mutation testing required less test cases than the coverage criteria. This is somehow contradictory to the results reported by other studies, i.e., Wong and Mathur [43] and Offutt *et al.* [30], which found that mutation requires more tests.

All these studies compared criteria-adequate test suites (test suites fulfilling all the requirements possessed by the criteria, i.e., killing all the mutants) and thus, their objectives were purely evaluating the effectiveness of the test criteria. However, while such an experimental design provides some insights regarding the test criteria, it makes the test effectiveness conclusion obscure as it is unclear whether test suites are better due to the inherent properties of the criteria or due to their sizes.

Other mutation-related studies that measure test effectiveness through hand-seeded faults are also some of those recorded on Table 1 (studies [3, 28, 37]). Andrews *et al.* [3] report that hand-seeded faults were much harder to detect than real faults, Papadakis and Malevris report on the application cost and fault detection capabilities of mutation testing strategies [37] and Namin and Kakarla report on the influence of test suite size, and programming language on test effectiveness [28].

2.3 Mutants and Defect Prediction

Recently researchers started using mutation as an indicator of error-proneness. Thus, they try to predict where (code components) the defects of the tested system are. Bowes *et al.* [6] demonstrated that using mutation score as a feature the accuracy of the defect prediction methods is improving.

Tengeri *et al.* [41] investigated whether statement coverage, mutation score, and reducibility (the amount of redundant test cases in the test suites, w.r.t. coverage and mutation) are good predictors of the expected number of remaining defects (confirmed defects, normalized by the system size). Their results show that coverage is not a good indicator, and that mutation and reducibility improve the predictions. Along the same lines, Ahmed *et al.* [1] measured the correlation between coverage, mutation score and subsequent bug-fix commits and found positive but weak correlations.

2.4 Test Suite Size and Test Effectiveness

Early research on software testing showed that test suite size is an important parameter influencing test effectiveness [12, 13]. The studies of Frankl *et al.* [12–14] consistently report that a positive relation between coverage attainment and fault detection exist, when test suite size is controlled. This is reported as non-linear and mainly appears at the highest coverage levels. Similarly, Chekam *et al.* [8] reports a positive relation but with insignificant, in practical terms, improvements (for coverage when test suite size is fixed). As these results investigated whether coverage provides enough guidance for improving test effectiveness, these findings do not concern the mutants and their representativeness (of real faults) in software testing experiments.

Namin and Andrews [27] investigated the role and influence of test suite size and coverage on test effectiveness, using mutants and real faults (13 faults for a single program) and concluded that both coverage and test suite size independently influence test effectiveness. Gligoric *et al.* [18] investigated the correlations between test suite size and mutation score and report mixed results, i.e., strong correlations for some programs, and weak for others.

Gopinath *et al.* [19] used regression analysis to model the relation between coverage and mutation score and report that test suite size did not improve the regression (indicating that test suite size does not contribute to explaining the variability). On the contrary, Inozemtseva and Holmes [21] report strong correlations between coverage and mutation score when ignoring test suite size, but weak correlations when test suite size is controlled. Therefore, concluding that test suite size is among the most important parameters that influence mutation score. Recently, Gay [17] investigated the fault detection capability (using 353 real faults) of different configurations of a search-based test generation tool (EvoSuite) and report that the test suite size had minor influence.

The studies of Andrews *et al.* [3] and Namin and Kakarla [28] also control for test suite size but found contradictory results (related to the underlying relation of mutation score and fault detection). Additionally, these two studies considered a single C program and a relatively small set of faults. In contrast, our study considers a large number of real faults from large real-world programs, written in both C and Java, and compares results related to the two main experimental evaluation metrics that are used in the literature.

Overall, from the above discussion, it should be obvious that there is much of controversy on the finding of previous studies and a poor understanding of the relation between test suite size and real fault detection. We expect a large empirical study can help clearing up some of this controversy.

3 EXPERIMENTAL PROCEDURE

3.1 Test Subjects

In our study, we use two sets of subjects, CoREBench [5] and Defects4J [23]. We choose these subjects as they form instances of relatively large projects that are accompanied by mature test suites as well as many real faults. CoREBench consists of four C programs named "Coreutils", "Findutils", "Grep" and "Make". Defects4J consists of five Java programs named "JFreeChart", "Closure", "Commons-Lang", "Commons-Math" and "Joda-Time".

Table 2: The subject programs used in the experiments. For each of them, their size in KLOC, the number of developer (Developer TC) and automatically generated test cases (Generated TC), and number of considered faults are presented.

Program	Description	Size	Developer TC	Generated TC	Faults
Coreutils	Utilities manipulating files and text	83	4,772	13,947	22
Findutils	Search directories utilities	18	1,054	3,877	15
Grep	Regular expression utilities	9	1,582	4,317	15
Make	Source code build system	35	528	163	18
JFreeChart	A chart library	96	3,691	111,279	19
Closure	Closure compiler	90	271,904	543,947	92
Commons-Lang	Java (library) - utilities	22	3,921	167,797	30
Commons-Math	Mathematics library	85	8,081	465,361	74
Joda-Time	Date and Time utilities	28	38,223	86,957	16

Details about the test subjects are recorded on Table 2. Both C and Java programs are of non-trivial size and all are accompanied by numerous developer test suites. All of the subjects are open source. The C subjects, "Coreutils", "Findutils", "Grep" and "Make" are tested by invoking them through command line, while the Java ones, "JFreeChart", "Closure", "Commons-Lang", "Commons-Math" and "Joda-Time", using the JUnit framework.

3.2 Fault Datasets (Defects4J and CoREBench)

Our study investigates the representativeness of mutants when they are used as test effectiveness evaluation metrics. We, therefore, need sets of typical instances of real faults that can be reliably reproduced by test cases. As we make a controlled experiment, we need projects with large and mature test suites so that we can adequately control and simulate our hypothesis.

Most of the previous studies rely on the programs from the Software Infrastructure Repository (SIR) [11, 20], typically using the programs composing the Siemens Suite, space and Unix utilities. Many of these programs includes artificially seeded faults and, consequently, are less relevant to this study, simply because we investigate the representativeness of mutants (which are artificially seeded faults themselves).

The space program in SIR is a notable exception as it comes with real faults. Yet, it is a single C program. Therefore, the degree to which any generalisation (to other programs and languages) is possible remains limited. For all these reasons, we consider SIR as less relevant for our study, and instead opt for benchmarks with multiple programs and real faults.

We, therefore, use two benchmarks with real faults, CoREBench and Defects4J, that have been systematically mined and isolated from the project source code repositories. These benchmarks have also been used (among many other studies) by the most relevant mutation-based studies, i.e., Just *et al.* [24] and Chekam *et al.* [8]. Therefore, our results complement the findings of these studies and can be compared and contrasted directly (whenever relevant).

CoREBench [5] is a collection of 70 fault instances of C programs, identified by test cases that reproduce reported faults (by writing test cases that reproduce the behaviour described in bug reports).

To identify the faults 4,000 commits were analysed by exercising them with the validating test cases in order to identify the fault-introducing and fault-fixing commits (test cases pass before the introduction of the fault, fail after and pass again after the fixing commit). Additional details about CoREBench can be found in the paper [5] and its accompanying website[1].

Defects4J [23] is a collection of 357 fault instances, in Java programs, identified by the developer test cases. The corresponding faulty and fixed versions were mined and manually verified by exercising them with the developer test cases. The differences (between the faulty and fixed version) were also manually refined (minimized so that they only include the faults) by removing unrelated changes such as program refactoring. Additional details about Defects4J can be found in the paper [23] and its GitHub webpage[2].

During our study, we verified the faulty and fixed versions using developer and automatically generated test suites. All faults have been isolated in single program instances and thus, similar to the previous studies [8, 24] we treat them as separate subjects. We excluded the CoREBench faults with identifiers 57 and 58 (from the Make program) due to technical issues, these versions failed to compile in the experimental environment. Similarly, we used 231 faults from Defects4J faults as the remaining 126 required infeasible amount of execution time for the experiment, i.e., more than an hour per test suite. As our analysis involves 21 test suites per studied fault, we were forced to adopt this rule in order to complete the experiment with reasonable resources.

3.3 Test Suites

CoREBench involves approximately 58,131 developer test cases. These test suites were later augmented (manually) by the studies of Böhme and Roychoudhury [5] (which added 70 test cases) and Chekam *et al.* [8] (which added 96 test cases) in order to exercise and detect the studied faults (designed by reproducing the burgs reported in bug reports) multiple times. Defects4J [23] includes approximately 553,477 developer tests.

As our experiment involves a uniform test suite selection and statistical analysis, we need a large, diverse and independently generated test cases. Therefore, we use the test pools created by Chekam *et al.* [8] (for the case of CoREBench) that are large, diverse and include multiple tests that exercise the faults in different ways. These include the developer test suites, the test cases generated with KLEE [33], in total 22,208 tests, and manually generated ones, 166 tests. Additional details about the test suites we used (for CoREBench) can be found in the work of Chekam *et al.* [8].

Defects4J [23] only includes the developer tests. Therefore, we augment these tests using two state-of-the-art test generation tools, Randoop [32] and EvoSuite [15]. Randoop was used for generating large numbers of random tests, whereas EvoSuite for generating large numbers of tests that maximise the branch coverage, weak mutation and strong mutation.

[1] http://www.comp.nus.edu.sg/~release/corebench/
[2] https://github.com/rjust/defects4j

We applied these tools five independent times with a relatively robust time budget per class (300 seconds), and limit the maximum number of tests to 2,000, per run. Overall, the process resulted in 20 test suites, composed of 1,375,341 test cases. Following the typical process for these experiments [23], all tests were automatically verified to ensure that they do not cause any compile errors, runtime errors and non-deterministic behaviour, using the available utilities provided by Defects4J [23].

3.4 Mutation Testing Tools

We employed the same tools as those used by the studies of Just *et al.* [24] and Chekam *et al.* [8]. We choose these tools as they are publicly available, robust and can produce results that are comparable to the existing work. We used these tools with the same settings as used in the initial studies.

Both tools support a set of commonly used mutant operators [25, 29], i.e., the AOR (Arithmetic Operator Replacement), LOR (Logical Operator Replacement), COR (Conditional Operator Replacement), ROR (Relational Operator Replacement), ORU (Operator Replacement Unary), STD (STatement Deletion), and LVR (Literal Value Replacement). Additional details about the mutation testing tools can be found in Just [22] as well as Chekam *et al.* [8].

3.5 Evaluation Metrics

We study the use of two evaluation metrics that are frequently employed by empirical studies. Such studies employ the following high level process:

(1) Create a large number of test suites, either by sampling test cases from a large pool of existing tests or by using a test generation algorithm or tool, until reaching a predetermined number of tests or criteria score level.
(2) Measure the criteria score such as coverage or mutation score (if test suites are size controlled) of the test suites or their size (if test suites are score controlled).
(3) Measure the fault detection capability of the test suites, by measuring either number or ratio of detected faults. Empirical studies usually employ isolated faulty versions (real faults) or hand-seeded faults or automatically seeded faults (mutants).
(4) Determine the test effectiveness based on one of the two following methods: 1) correlation analysis between fault detection ratios and criteria scores such as coverage or mutant detection ratios (*correlation method*), or 2) fault detection ratios at predefined score points such as the highest criteria score levels achieved by the available test suites (*fault detection method*).

The correlation method is becoming increasingly popular and is used to judge the effectiveness of the test methods. It is also frequently used to perform the effectiveness comparison of the test techniques, e.g., adopted by studies with references [18, 21, 44] perform such effectiveness comparisons.

The fault detection method simply compares the fault detection capabilities of test suites at a score level of interest (usually at 100% or close to 100% score levels), e.g., adopted by studies with references [12, 26, 43].

The two evaluation metrics, i.e., correlation coefficients (used in the correlation method) and fault detection ratios (used in the fault detection method), are often being confused as being the same. However, in practice they capture different aspects of the test criteria and should be investigated distinctly. In the present study, we demonstrate that mutants can correlate weakly with fault detection, but they can provide statistically and practically significant fault detection improvements over randomly selected test suites (or test suites with lower scores) when reaching higher mutation score levels.

3.6 Data Analysis

Initially, we form our test pools by merging the automatically generated with the developer test cases. We, then, sample 10,000 test suites of random sizes in the range (0-20% of the size of the test pool). Then, we categorise these suites as 'Failing', i.e., they involve at least one test case fails, and 'Passing', i.e., all test cases pass, and plot their mutation scores and test suite sizes respectively. This visualisation gives us an initial indication on the examined relations. We then perform regression analysis (Logistic regression) on these data by modelling the relationships between test suite size, mutation score, combination of test suite size and mutation score, and fault detection. By inspecting the resulting p–values and pseudo R^2 values, we investigate which of these parameters play the most important role and measure the fit of the models.

To further analyse our data and replicate previous findings, we apply both the correlation and the fault detection methods. For the correlation method, we use the Kendall and Pearson coefficients to compute the correlation between Mutation Score (MS) and Fault Detection (FD) on the 10,000 suites of random sizes. The resulting coefficients show the association between MS and FD when the test suite size is not controlled. For size controlled results, we sample test suites of the same size (without replacement) for different test suite size groups (increments of 2.5% in the range 0-50% of the test pool size). Thus, we sample 10,000 test suites per size group and compute the correlations between MS and FD. These results show the correlations independent of test suite size.

For the fault detection method, we order the test suites, for every selected size, according to their mutation score and create three sets of test suites: those that are ranked within top 25%, top 10% and all of them. We then perform a Chi-squared test, which is a non-parametric proportion test, to compare the fault detection of the pairs (top 25% and all test suites, top 10% and all test suites) and compute the confidence intervals for 95 percent. These values represent the fault detection probabilities of the top ranked suites compared with the whole set of test suites (baseline).

4 RESULTS

4.1 Visualisations

Figure 1 shows the test suite size and mutation score values of the failing and passing test suites. As can be seen from the boxplots, the trend is that failing test suites are of larger sizes and, at the same time, achieve higher mutation scores than the passing ones. This indicates that both variables may influence the fault detection ability of the test suites.

To further explore the interconnection between size and mutation score, we investigate the relation between test suite size and mutation score. Due to space constraints, we do not show this plot, however, the results clearly indicate a logarithmic relation, i.e., $y = ax^b$ where $(s < b < 1)$. Therefore, we expect that test suite size and mutation score are interconnected in such a way that higher (or lower) mutation score levels require increased (or decreased) number of test cases. This leads to the question of whether test suite size or mutation score explain the test effectiveness.

4.2 Regression Models

To get an insight on the relation between test suite size and fault detection, we apply regression analysis. As our data involve a continuous independent variable (MS: mutation score), a discrete independent variable ($Size$: test suite size), and a binary dependent variable (FD: fault detection) we apply a logistic regression. We examined various models between these variables, i.e., $Size * MS$, MS, $Size$, and determined the p–values indicating their significance.

The output of the regression indicates that all the examined variables associate to fault detection (test suite size, mutation score, and their combination). They are significantly associated with the probability of detecting faults (p–values < 0.05). Therefore, the results suggest that all examined variables independently influence test effectiveness.

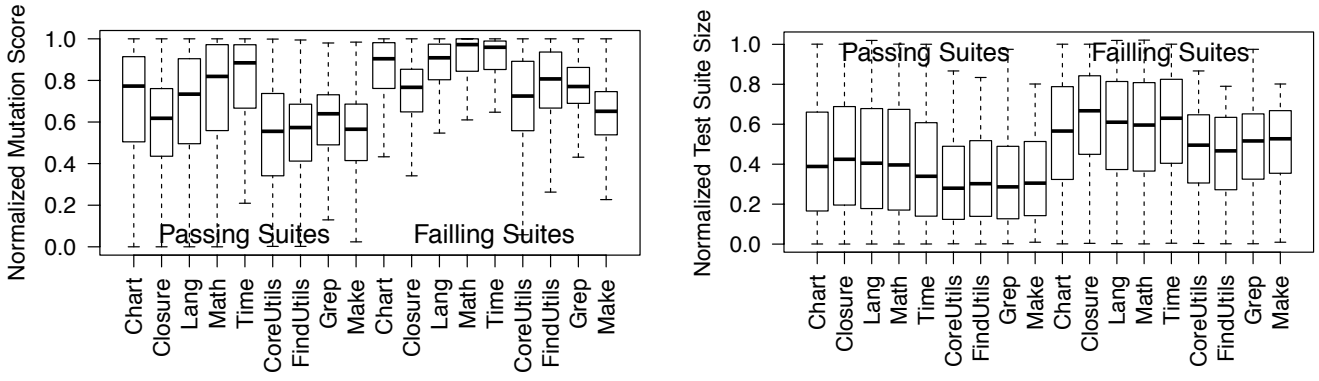
To evaluate the fit of the regression model we calculated the pseudo R^2 values of the models. Figure 2 present the resulting pseudo R^2 values. These data indicate that both the models of test suite size and mutation scores (data with labels Size and MS) are similar. Size models have a higher predictive power (and less variation) than the mutation score models in Defects4J while lower in CoreBench. The combined model ($Size * MS$) outperforms both Size and MS models, in terms of their predictive power. The differences are statistically significant (determined by using a Wilcoxon signed-rank test, and significance level 0.001) suggesting that both Size, MS and their combination contributes to test effectiveness. However, the majority of the predictions is of moderate strengths.

One first finding is that the fault detection cannot be predicted well by mutation scores, size or their combination.

4.3 Correlation Analysis with Test Suite Size Controlled/Uncontrolled

To investigate the extent of confounding effects of test suite size, we perform a correlation analysis by controlling (holding test suite size constant) and not controlling (selecting test suites of arbitrary sizes) for test suites size. This is a typical process (also followed by existing work [8, 21, 28]) that eliminates the influence of the independent variable (size) on the observed effect and helps determine whether mutation score is associated with fault detection (and the strength of this association).

Our analysis is based on the Kendall and Pearson correlation coefficients. We measure the correlations for various constant test suite sizes, sizes in the range 2.5%-50%, of the test pools, incremented by 2.5% and for arbitrary selected test suites of the same size: Figure 3 shows the results. The correlations on the left side of the figure regard the uncontrolled size case while on the right side of the figure the controlled one.



(a) Mutation Score Vs Fault Detection.

(b) Test suite size Vs Fault Detection.

Figure 1: Mutation Score and Test suite size of the Passing and Failing Test Suites. Failing Test Suites have higher mutation scores and suite sizes than the Passing ones.

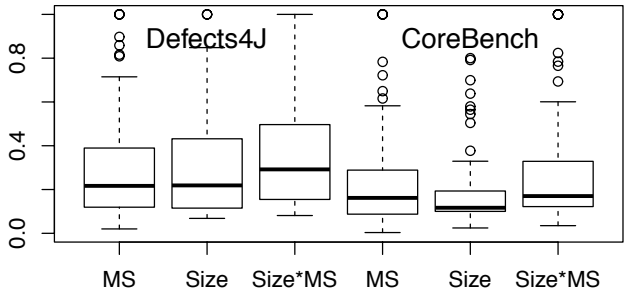


Figure 2: Regression Models (R^2 values) for Mutation Score (MS), Test Suite Size (Size) and their Combination (Size*MS). All p–values are significant.

The results of the uncontrolled size case show moderate to strong correlations (the majority of the values is within the range 0.35 to 0.75). These correlations become relatively weak (approximately within the range 0.05 to 0.20) when the suite size is controlled (i.e., the influence of size disappears). These results suggest that a major part of the association between mutation score and fault detection is simply an effect of size. Still a positive relation between them exists, but it is relatively weak.

Interestingly, our results are fairly consistent across different programs and the different programming languages (Java and C) we used. This adds to the evidence that the results we observe are valid. They also suggest that mutants have a consistent behaviour among the subjects we used.

Finally, it is worth noting that our results (regarding the uncontrolled test size case) are consistent with those reported by the work of Just *et al.* [24], which did not consider the test size effects. This fact provides confidence on our analysis. However, in contrast to the conclusion of Just *et al.* [24], our results demonstrate the effect of size and overall that mutants are not strongly correlated with fault detection, when size is controlled.

4.4 Fault Detection Probabilities

Mutation testing is known for its test effectiveness [8, 16, 31], while our analysis shows that mutation score is not (strongly) correlated with fault detection. Does this mean that mutation testing is significantly overrated? While this is a possibility, another explanation could be that mutation testing is helpful at guiding testers to improve the fault detection capabilities of test suites, while it is not that good at representing the behaviour of the real faults. It could be that the relation is non-linear and improves only at the highest mutation score levels as suggested by some previous studies [12].

To investigate this, for every selected size of test suites, we compute the fault detection probabilities (using the Chi-squared test) of the test suites with higher mutation scores and compare them with the fault detection probabilities of all randomly selected test suites of the same size. In this case, any differences we may observe (on the fault detection probabilities) would be attributed to the mutation score differences and as we keep the test suite size constant they are independent of the size effects.

Such a process was followed by the recent study of Frankl and Iakounenko [12], and reports that test suites with higher coverage scores reveal a significantly higher number of faults. As in our case, correlations are weak, what happens to the fault detection at highest mutation score levels? We hypothesise that fault detection probability can be important given the results reported by the studies of Chekam *et al.* [8] and Frankl with her colleagues [12–14] that reports a significant fault detection only at the highest score levels (while not much of difference at the majority of the scores).

Figure 4 shows the improvement on the fault detection probabilities between the 10% of test suites (with the highest mutation scores) and randomly selected test suites of the same size. The points in the plot represent the statistically significant improvements (at 0.05 significance level) on the probabilities and their confidence intervals (for 95 percent confidence level) computed using the Chi-squared test. The Defects4J faults that have statistically significant improvements are 138, while the CoreBench are 59. This suggests that we have no evidence to support the claim that there is a significant improvement for the 40% (93 out of 231) and 13% (9 out of 68) of the Defects4J and CoreBench faults.

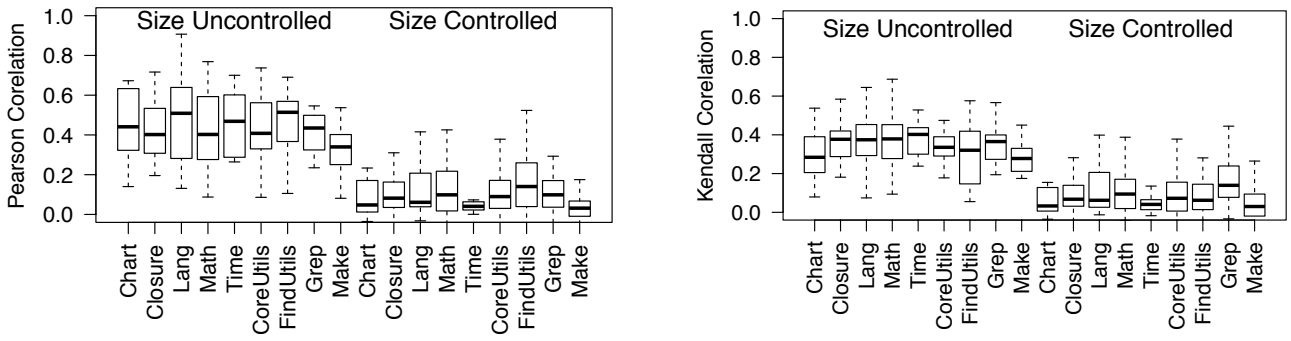


Figure 3: Correlation between mutation score and fault detection. Correlations are relatively strong when test suite size is uncontrolled but drop significantly when test suite size is controlled.

Interestingly, despite the weak correlations, the fault detection improvements (for the statistically significant cases) can be considered of practical significance as the majority of the points are approximately 10% for Defects4J and above 30% for CoreBench. Specifically, the average improvements on fault detection for the top ranked 25% and 10% of the test suites are 8% and 11% for Defects4J and 18% and 46% for CoreBench. These results indicate that mutation is indeed strong and can successfully guide testers towards improving their test suites. Unfortunately, as also indicated by the related work, testers have to reach a relatively high score level before they can be confident on their testing [8].

Our results reveal that the correlation method and the fault detection method capture different aspects of test effectiveness. This implies that future empirical studies that use these evaluation methods should interpret them accordingly. We discuss this issue in detail in Section 5.2.

5 DISCUSSION

Our results show that the correlations of mutation score and real faults are significant, but notably weaker than reported in the literature and assumed by empirical studies. This finding is in essence negative as mutants seem to be relatively unreliable substitutes of real faults (as a means to support controlled experiments). Despite this, we found significant improvements in fault detection (of test suites at the highest score levels) suggesting that mutants can provide valuable guidance and establish confidence.

In the remainder of this section, we try to shed some light on the mutant-fault relation, by investigating the behavioural similarities of mutants and real faults. We then discuss possible implications of the practical differences of using correlation analysis and fault detection methods in software testing experiments.

5.1 Behavioural Similarity between Mutants and Real Faults

Are mutants a valid substitute of real faults in controlled experiments? If we hypothesise that they are, then we should expect that the majority of the mutants behaves similarly to the real faults. In a sense, we expect that when faults are detected, the majority of the mutants are killed, and when not, the majority remain live.

To investigate this, we measure the behaviour similarity of every single mutant and the respective faults. To do so, we use a typical similarity coefficient, named Ochiai[3], which we borrow from fault localisation studies [42]). Ochiai takes values in the range $[0, 1]$ indicating the similarity level, with 0 indicating completely different and 1 exactly the same.

Figure 5 records the maximum similarity and the similarity of all the mutants for every considered faults. As can be seen from the maximum similarity values, the majority of the faults is simulated very well by at least one mutant, the one with the maximum similarity (from the left box-plot we can observe that 50% of the data have similarities of at least 90%), while the great majority of the mutants behave differently. This implies that some mutants simulate well the behaviour of the faults, while the majority of them do not (we measure approximately 1% of all mutants have behaviour similarities above 0.5).

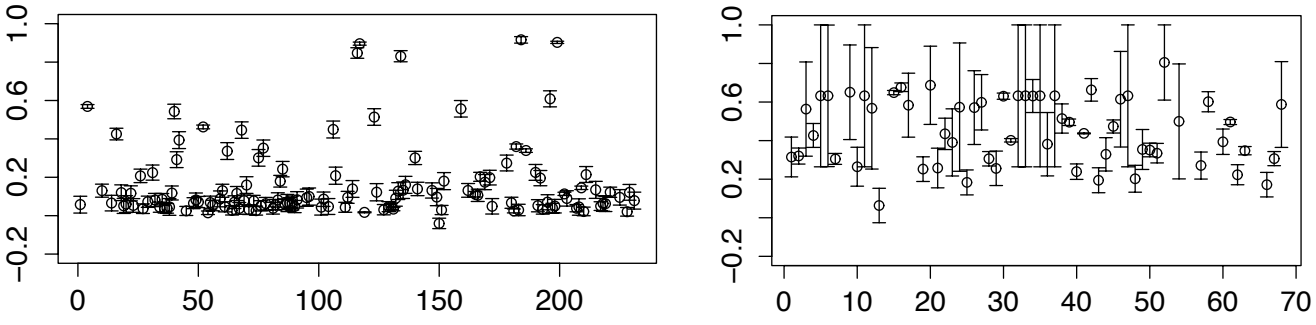
Overall, our results reveal that irrelevant mutants cause the weak correlations. We, therefore, believe, that future research should focus on identifying mutants that are linked with the faults.

5.2 Correlations vs. Fault Detection

To investigate the practical differences between the evaluation methods, i.e., the correlation and the actual fault detection, we measure the association between the two evaluation metrics, i.e., correlation coefficients and fault detection improvement. Informally, we check whether high/low correlation coefficients imply high/low fault detection improvement. A strong association will indicate that one method implies the other, while a weak association that the two methods capture different aspects of interest.

For every fault in our dataset, we measure a) the correlation coefficients (between mutation score and fault detection) and b) the improvement on fault detection (approximated by the fault detection rate differences of the top ranked test suites and all selected test suites of the same size). Our raw data, when using the Pearson correlation and fault detection improvements at the top 10%, are depicted on Figure 6. To further investigate the association between a) and b), we used Kendall and Pearson correlation on the data where we had statistically significant fault detection improvements.

[3]The Ochiai coefficient measures the similarity between two variables given a set of observations, it is equivalent to the cosine similarity.



(a) Defects4J (b) CoreBench

Figure 4: Improvement on the fault detection probabilities (intervals) with test suite size controlled. The values represent the difference on the fault detection probabilities between the test suites with the highest mutation score and randomly selected ones (top 10% of test suites Vs randomly selected). We observe that mutants lead to significant fault detection improvements.

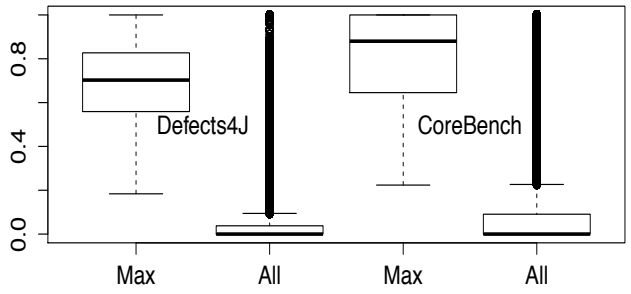


Figure 5: Behaviour Similarity between Mutants and Real Faults. Most of the faults' behaviour is simulated well by the (few) mutants with maximum similarity, but not by the majority of the mutants (similarity of all mutants).

Our results show a moderate to strong association between the two metrics. In particular, the association between the correlation coefficients, Kendall and Pearson (used to measure the correlation between mutation score and fault detection), with the fault detection improvement of the top 25% of test suites was 0.621 and 0.504 when using the Kendall's rank correlation coefficient τ and 0.860 and 0.732 when using the Pearson correlation (for Defects4J and CoreBench respectively). The Kendall's rank correlations were found to be 0.471 and 0.500 and the Pearson correlations were found to be 0.673 and 0.652, respectively for Defects4J and CoreBench. Interestingly, by comparing the correlations of the test suites scoring at the top 25% and 10%, we observe that the disagreement between the two metrics is increasing when moving at higher score levels.

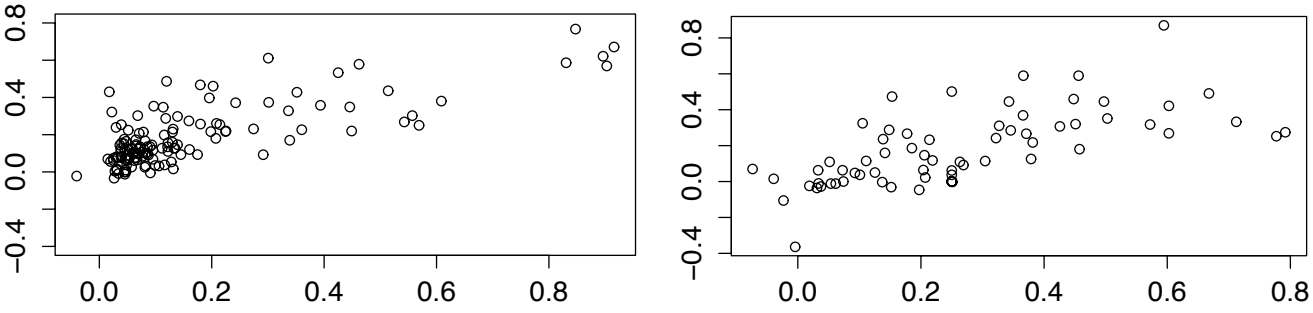
Overall, since the association is moderate, we can conclude that correlations are good at providing trends, but at the same time they do not capture the actual improvements at the specific points of interest (such as the fault detection improvement at the highest score levels).

6 THREATS TO VALIDITY

The most important question in empirical studies is the extent to which their findings generalise. To cater for this issue, we performed the largest empirical study to date, using two fault sets from independent studies, written in C and Java, and found consistent results across all subjects. However, we acknowledge that since these programs are open source and the faults were mined from source code repositories, our results might not be representative of actual industrial systems and other "pre-release" faults (faults not recorded in the repositories).

Another threat may be due to the incompleteness of the faults in the benchmarks we study. As the fault sets are not exhaustive, we can only assume that some code parts are faulty. This means that we cannot assume that the rest of the code is not faulty. Thus, our results might be subject to noise caused by mutants residing on code parts that are irrelevant, to the studied faults, but are relevant to other 'unknown' faults. To diminish this threat, we performed our analysis using only relevant, to the studied faults, test cases (tests with dependence to the faulty component). To further check our results, we also examined a sample of 10 faults from the CoreUtils by considering only the mutants that are residing on the faulty and directly dependent (control data dependencies) statements, to the faulty statements, and found very similar (weak) correlations. Overall, we are confident that the above issues are not of particular importance since we replicate (to some extent) previous studies and find consistent trends among all the considered subjects.

Threats may arise from the automatically augmented test suites we used. While the augmented test suites may not be representative of tests generated entirely by human engineers, the augmentation was necessary so that we could sample multiple, size-controlled test suites from a large, strong, and diverse test pool. To mitigate the threat from augmentation, we repeated our experiments without the augmentation (i.e., using either only the developer test suites or only the automatically generated test suites) and found exactly the same trends (with slightly weaker correlations and fault detection improvements).



(a) Defects4J (b) CoreBench

Figure 6: Pearson correlation coefficients Vs fault detection improvement.

Similarly, due to technical difficulties, we did not analyse 126 faults from Defect4J. To reduce this threat, we measured the correlations when using (only) the developer test suites for all but 5 faults[4] of the benchmark and found similar results.

Other threats might be attributed to the way we handled equivalent mutants, which was based on the composed test pools. Though, we used state of the art test generation tools and overall achieved a relatively high mutation score. Unfortunately, there are clear limits to any practical identification of equivalent mutants, as the problem itself is undecidable. While we simply admit that this is a potential threat to our study, we would argue that, in practice, there is not much else that can be done (at least by today's standards).

There may be threats due to the tools we used. Though, these tools were developed by independent studies [8, 22]. To cater for this threat, we also used Pit [9], a Java mutation testing tool that is quite popular in mutation testing studies [25, 36]. We repeated our analysis with Pit using two test suites of Randoop and EvoSuite and found similar results, i.e., the correlations and fault detection improvements did not differ statistically. Therefore, we are confident that our results generalise to Pit as well.

Following the lines of previous work [4, 22, 28], we applied all of our analysis on the fixed program version version. Although this is a common practice, our results might not hold on the cases of faulty program versions [8]. Though, we were forced to do so, in order to replicate the previous studies. However, we believe that this threat is not of actual importance as we are concerned with mutation testing, which according had a small variation on mutant couplings of the fixed the faulty programs [8].

Finally, we used Kendall and Pearson correlation coefficients to measure the association between the studied variables, while the work of Just *et al.* [24] used the Biserial correlation. Unfortunately, Biserial correlation assumes equal variance between the instances of the dichotomous variable under analysis, which we found inapplicable in our case as the difficulty of detecting faults differs. Nevertheless, we also computed the Biserial correlations and found the same results (the differences were smaller than 0.001). We therefore do not consider this as a critical threat.

[4] 5 faults were not considered as they required exceedingly long time to complete (more than an hour per test case).

7 CONCLUSIONS

Our main conclusion is that both test suite size and mutation score influence fault detection. Our results support the claim that relatively strong correlations between mutation score and fault detection exist, as suggested by previous work [24]. However, these are simply a product of the dependence between mutation score and test suite size. Our data show that when controlling the test suite size (decouple mutation score from test suite size) all correlations become weak or moderate in the best case.

In practice, our results suggest that using mutants as substitutes of real faults (as performed in most of the software testing experiments) can be problematic. Despite the weak correlations we observe, our results also show that the fault detection improves significantly when test suites reach the highest mutation score levels. Taken together, our findings suggest that mutation score is actually helpful, to testers for improving test suites (by reaching relative high levels of mutation scores), but it is not that good at representing the actual test effectiveness (real fault detection).

A potential explanation of this point concerns the "nature" of mutants and the real faults. Mutants are generated following a systematic procedure, while real faults are the specific instances that escape programmer's attention. Thus, mutants represent a large number and variety of possible faulty instances, while faults are by nature few and irrelevant to the majority of the mutants.

We also found that some mutants are indeed capable of representing the behaviour of real faults for most of our subjects. However, as the majority of the mutants involved have no representative behaviour, the mutation scores are inflated and the correlations are weak. This finding suggests that future research should focus on generating more representative sets of mutants.

ACKNOWLEDGEMENTS

This research was supported by the Next-Generation Information Computing Development Program of the Korean National Research Foundation (NRF), funded by the Ministry of Science, ICT (No. 2017M3C4A7068179), and the Institute for Information & communications Technology Promotion (IITP) grant, funded by the Korea government (MSIP) (No.R0126-18-1101, (SW Star Lab) Software R&D for Model-based Analysis and Verification of Higher-order Large Complex System).

REFERENCES

[1] Iftekhar Ahmed, Rahul Gopinath, Caius Brindescu, Alex Groce, and Carlos Jensen. 2016. Can testedness be effectively measured?. In *Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE 2016, Seattle, WA, USA, November 13-18, 2016*. 547–558. https://doi.org/10.1145/2950290.2950324

[2] Paul Ammann and Jeff Offutt. 2008. *Introduction to software testing*. Cambridge University Press.

[3] James H. Andrews, Lionel C. Briand, and Yvan Labiche. 2005. Is mutation an appropriate tool for testing experiments?. In *27th International Conference on Software Engineering (ICSE 2005), 15-21 May 2005, St. Louis, Missouri, USA*. 402–411. https://doi.org/10.1145/1062455.1062530

[4] James H. Andrews, Lionel C. Briand, Yvan Labiche, and Akbar Siami Namin. 2006. Using Mutation Analysis for Assessing and Comparing Testing Coverage Criteria. *IEEE Trans. Software Eng.* 32, 8 (2006), 608–624. https://doi.org/10.1109/TSE.2006.83

[5] Marcel Böhme and Abhik Roychoudhury. 2014. CoREBench: studying complexity of regression errors. In *International Symposium on Software Testing and Analysis, ISSTA '14, San Jose, CA, USA - July 21 - 26, 2014*. 105–115. https://doi.org/10.1145/2610384.2628058

[6] David Bowes, Tracy Hall, Mark Harman, Yue Jia, Federica Sarro, and Fan Wu. 2016. Mutation-aware fault prediction. In *Proceedings of the 25th International Symposium on Software Testing and Analysis, ISSTA 2016, Saarbrücken, Germany, July 18-20, 2016*. 330–341. https://doi.org/10.1145/2931037.2931039

[7] Cristian Cadar, Daniel Dunbar, and Dawson R. Engler. 2008. KLEE: Unassisted and Automatic Generation of High-Coverage Tests for Complex Systems Programs. In *8th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2008, December 8-10, 2008, San Diego, California, USA, Proceedings*. 209–224. http://www.usenix.org/events/osdi08/tech/full_papers/cadar/cadar.pdf

[8] Thierry Titcheu Chekam, Mike Papadakis, Yves Le Traon, and Mark Harman. 2017. An empirical study on mutation, statement and branch coverage fault revelation that avoids the unreliable clean program assumption. In *Proceedings of the 39th International Conference on Software Engineering, ICSE 2017, Buenos Aires, Argentina, May 20-28, 2017*. 597–608. https://doi.org/10.1109/ICSE.2017.61

[9] Henry Coles, Thomas Laurent, Christopher Henard, Mike Papadakis, and Anthony Ventresque. 2016. PIT: a practical mutation testing tool for Java (demo). In *Proceedings of the 25th International Symposium on Software Testing and Analysis, ISSTA 2016, Saarbrücken, Germany, July 18-20, 2016*. 449–452. https://doi.org/10.1145/2931037.2948707

[10] Muriel Daran and Pascale Thévenod-Fosse. 1996. Software Error Analysis: A Real Case Study Involving Real Faults and Mutations. In *Proceedings of the 1996 International Symposium on Software Testing and Analysis, ISSTA 1996, San Diego, CA, USA, January 8-10, 1996*. 158–171. https://doi.org/10.1145/229000.226313

[11] Hyunsook Do, Sebastian G. Elbaum, and Gregg Rothermel. 2005. Supporting Controlled Experimentation with Testing Techniques: An Infrastructure and its Potential Impact. *Empirical Software Engineering* 10, 4 (2005), 405–435. https://doi.org/10.1007/s10664-005-3861-2

[12] Phyllis G. Frankl and Oleg Iakounenko. 1998. Further Empirical Studies of Test Effectiveness. In *SIGSOFT '98, Proceedings of the ACM SIGSOFT International Symposium on Foundations of Software Engineering, Lake Buena Vista, Florida, USA, November 3-5, 1998*. 153–162. https://doi.org/10.1145/288195.288298

[13] Phyllis G. Frankl and Stewart N. Weiss. 1993. An Experimental Comparison of the Effectiveness of Branch Testing and Data Flow Testing. *IEEE Trans. Software Eng.* 19, 8 (1993), 774–787. https://doi.org/10.1109/32.238581

[14] Phyllis G. Frankl, Stewart N. Weiss, and Cang Hu. 1997. All-uses vs mutation testing: An experimental comparison of effectiveness. *Journal of Systems and Software* 38, 3 (1997), 235–253. https://doi.org/10.1016/S0164-1212(96)00154-9

[15] Gordon Fraser and Andrea Arcuri. 2011. EvoSuite: automatic test suite generation for object-oriented software. In *SIGSOFT/FSE'11 19th ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE-19) and ESEC'11: 13th European Software Engineering Conference (ESEC-13), Szeged, Hungary, September 5-9, 2011*. 416–419. https://doi.org/10.1145/2025113.2025179

[16] Gordon Fraser and Andreas Zeller. 2012. Mutation-Driven Generation of Unit Tests and Oracles. *IEEE Trans. Software Eng.* 38, 2 (2012), 278–292. https://doi.org/10.1109/TSE.2011.93

[17] Gregory Gay. 2017. The Fitness Function for the Job: Search-Based Generation of Test Suites That Detect Real Faults. In *2017 IEEE International Conference on Software Testing, Verification and Validation, ICST 2017, Tokyo, Japan, March 13-17, 2017*. 345–355. https://doi.org/10.1109/ICST.2017.38

[18] Milos Gligoric, Alex Groce, Chaoqiang Zhang, Rohan Sharma, Mohammad Amin Alipour, and Darko Marinov. 2015. Guidelines for Coverage-Based Comparisons of Non-Adequate Test Suites. *ACM Trans. Softw. Eng. Methodol.* 24, 4 (2015), 22:1–22:33. https://doi.org/10.1145/2660767

[19] Rahul Gopinath, Carlos Jensen, and Alex Groce. 2014. Code coverage for suite evaluation by developers. In *36th International Conference on Software Engineering, ICSE '14, Hyderabad, India - May 31 - June 07, 2014*. 72–82. https://doi.org/10.1145/2568225.2568278

[20] Monica Hutchins, Herbert Foster, Tarak Goradia, and Thomas J. Ostrand. 1994. Experiments of the Effectiveness of Dataflow- and Controlflow-Based Test Adequacy Criteria. In *Proceedings of the 16th International Conference on Software Engineering*. 191–200. http://portal.acm.org/citation.cfm?id=257734.257766

[21] Laura Inozemtseva and Reid Holmes. 2014. Coverage is not strongly correlated with test suite effectiveness. In *36th International Conference on Software Engineering, ICSE '14, Hyderabad, India - May 31 - June 07, 2014*. 435–445. https://doi.org/10.1145/2568225.2568271

[22] René Just. 2014. The major mutation framework: efficient and scalable mutation analysis for Java. In *International Symposium on Software Testing and Analysis, ISSTA '14, San Jose, CA, USA - July 21 - 26, 2014*. 433–436. https://doi.org/10.1145/2610384.2628053

[23] René Just, Darioush Jalali, and Michael D. Ernst. 2014. Defects4J: a database of existing faults to enable controlled testing studies for Java programs. In *International Symposium on Software Testing and Analysis, ISSTA '14, San Jose, CA, USA - July 21 - 26, 2014*. 437–440. https://doi.org/10.1145/2610384.2628055

[24] René Just, Darioush Jalali, Laura Inozemtseva, Michael D. Ernst, Reid Holmes, and Gordon Fraser. 2014. Are mutants a valid substitute for real faults in software testing?. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, (FSE-22), Hong Kong, China, November 16 - 22, 2014*. 654–665. https://doi.org/10.1145/2635868.2635929

[25] Marinos Kintis, Mike Papadakis, Andreas Papadopoulos, Evangelos Valvis, Nicos Malevris, and Yves Le Traon. 2017. How effective are mutation testing tools? An empirical analysis of Java mutation testing tools with manual analysis and real faults. *Empirical Software Engineering* (21 Dec 2017). https://doi.org/10.1007/s10664-017-9582-5

[26] Nan Li, Upsorn Praphamontripong, and Jeff Offutt. 2009. An Experimental Comparison of Four Unit Test Criteria: Mutation, Edge-Pair, All-Uses and Prime Path Coverage. In *Mutation 2009, Denver, Colorado, USA*. 220–229. https://doi.org/10.1109/ICSTW.2009.30

[27] Akbar Siami Namin and James H. Andrews. 2009. The influence of size and coverage on test suite effectiveness. In *Proceedings of the Eighteenth International Symposium on Software Testing and Analysis, ISSTA 2009, Chicago, IL, USA, July 19-23, 2009*. 57–68. https://doi.org/10.1145/1572272.1572280

[28] Akbar Siami Namin and Sahitya Kakarla. 2011. The use of mutation in testing experiments and its sensitivity to external threats. In *Proceedings of the 20th International Symposium on Software Testing and Analysis, ISSTA 2011, Toronto, ON, Canada, July 17-21, 2011*. 342–352. https://doi.org/10.1145/2001420.2001461

[29] A. Jefferson Offutt, Ammei Lee, Gregg Rothermel, Roland H. Untch, and Christian Zapf. 1996. An Experimental Determination of Sufficient Mutant Operators. *ACM Trans. Softw. Eng. Methodol.* 5, 2 (1996), 99–118. https://doi.org/10.1145/227607.227610

[30] A. Jefferson Offutt, Jie Pan, Kanupriya Tewary, and Tong Zhang. 1996. An Experimental Evaluation of Data Flow and Mutation Testing. *Softw., Pract. Exper.* 26, 2 (1996), 165–176. https://doi.org/10.1002/(SICI)1097-024X(199602)26:2<165::AID-SPE5>3.0.CO;2-K

[31] Jeff Offutt. 2011. A mutation carol: Past, present and future. *Information & Software Technology* 53, 10 (2011), 1098–1107. https://doi.org/10.1016/j.infsof.2011.03.007

[32] Carlos Pacheco and Michael D. Ernst. 2007. Randoop: feedback-directed random testing for Java. In *Companion to the 22nd Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2007, October 21-25, 2007, Montreal, Quebec, Canada*. 815–816. https://doi.org/10.1145/1297846.1297902

[33] Hristina Palikareva, Tomasz Kuchta, and Cristian Cadar. 2016. Shadow of a doubt: testing for divergences between software versions. In *Proceedings of the 38th International Conference on Software Engineering, ICSE 2016, Austin, TX, USA, May 14-22, 2016*. 1181–1192. https://doi.org/10.1145/2884781.2884845

[34] Mike Papadakis, Christopher Henard, Mark Harman, Yue Jia, and Yves Le Traon. 2016. Threats to the validity of mutation-based test assessment. In *Proceedings of the 25th International Symposium on Software Testing and Analysis, ISSTA 2016, Saarbrücken, Germany, July 18-20, 2016*. 354–365. https://doi.org/10.1145/2931037.2931040

[35] Mike Papadakis, Yue Jia, Mark Harman, and Yves Le Traon. 2015. Trivial Compiler Equivalence: A Large Scale Empirical Study of a Simple, Fast and Effective Equivalent Mutant Detection Technique. In *37th IEEE/ACM International Conference on Software Engineering, ICSE 2015, Florence, Italy, May 16-24, 2015, Volume 1*. 936–946. https://doi.org/10.1109/ICSE.2015.103

[36] Mike Papadakis, Marinos Kintis, Jie Zhang, Yue Jia, Yves Le Traon, and Mark Harman. 2018. Mutation Testing Advances: An Analysis and Survey. *Advances in Computers* (2018).

[37] Mike Papadakis and Nicos Malevris. 2010. An Empirical Evaluation of the First and Second Order Mutation Testing Strategies. In *Third International Conference on Software Testing, Verification and Validation, ICST 2010, Paris, France, April 7-9, 2010, Workshops Proceedings*. 90–99. https://doi.org/10.1109/ICSTW.2010.50

[38] Rudolf Ramler, Thomas Wetzlmaier, and Claus Klammer. 2017. An empirical study on the application of mutation testing for a safety-critical industrial software system. In *Proceedings of the Symposium on Applied Computing, SAC 2017, Marrakech,*

Morocco, April 3-7, 2017. 1401–1408. https://doi.org/10.1145/3019612.3019830

[39] David Schuler and Andreas Zeller. 2013. Covering and Uncovering Equivalent Mutants. *Softw. Test., Verif. Reliab.* 23, 5 (2013), 353–374. https://doi.org/10.1002/stvr.1473

[40] Donghwan Shin, Shin Yoo, and Doo-Hwan Bae. 2017. A Theoretical and Empirical Study of Diversity-aware Mutation Adequacy Criterion. *IEEE Trans. Software Eng.* (2017). https://doi.org/10.1109/TSE.2017.2732347

[41] Dávid Tengeri, László Vidács, Árpád Beszédes, Judit Jász, Gergo Balogh, Bela Vancsics, and Tibor Gyimóthy. 2016. Relating Code Coverage, Mutation Score and Test Suite Reducibility to Defect Density. In *Ninth IEEE International Conference on Software Testing, Verification and Validation Workshops, ICST Workshops 2016, Chicago, IL, USA, April 11-15, 2016*. 174–179. https://doi.org/10.1109/ICSTW.2016.25

[42] W. Eric Wong, Ruizhi Gao, Yihao Li, Rui Abreu, and Franz Wotawa. 2016. A Survey on Software Fault Localization. *IEEE Trans. Software Eng.* 42, 8 (2016), 707–740. https://doi.org/10.1109/TSE.2016.2521368

[43] W. Eric Wong and Aditya P. Mathur. 1995. Fault detection effectiveness of mutation and data flow testing. *Software Quality Journal* 4, 1 (1995), 69–83. https://doi.org/10.1007/BF00404650

[44] Yucheng Zhang and Ali Mesbah. 2015. Assertions are strongly correlated with test suite effectiveness. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2015, Bergamo, Italy, August 30 - September 4, 2015*. 214–224. https://doi.org/10.1145/2786805.2786858