Assertions Are Strongly Correlated with Test Suite Effectiveness

Yucheng Zhang
Electrical and Computer Engineering
University of British Columbia
Vancouver, BC, Canada
yuchengz@ece.ubc.ca

Ali Mesbah
Electrical and Computer Engineering
University of British Columbia
Vancouver, BC, Canada
amesbah@ece.ubc.ca

ABSTRACT

Code coverage is a popular test adequacy criterion in practice. Code coverage, however, remains controversial as there is a lack of coherent empirical evidence for its relation with test suite effectiveness. More recently, test suite size has been shown to be highly correlated with effectiveness. However, previous studies treat test methods as the smallest unit of interest, and ignore potential factors influencing this relationship. We propose to go beyond test suite size, by investigating test assertions inside test methods. We empirically evaluate the relationship between a test suite's effectiveness and the (1) number of assertions, (2) assertion coverage, and (3) different types of assertions. We compose 6,700 test suites in total, using 24,000 assertions of five real-world Java projects. We find that the number of assertions in a test suite strongly correlates with its effectiveness, and this factor directly influences the relationship between test suite size and effectiveness. Our results also indicate that assertion coverage is strongly correlated with effectiveness and different types of assertions can influence the effectiveness of their containing test suites.

Categories and Subject Descriptors

D.2.5 [**Software Engineering**]: Testing and Debugging; D.2.8 [**Software Engineering**]: Metrics

General Terms

Experimentation, Measurement

Keywords

Test suite effectiveness; assertions; coverage;

1. INTRODUCTION

Software testing has become an integral part of software development. A software product cannot be confidently released unless it is adequately tested. Code coverage is the most popular test adequacy criterion in practice. However, coverage alone is not the goal of software testing, since coverage without checking for correctness is meaningless. A more meaningful adequacy metric is the fault detection ability of a test suite, also known as test suite effectiveness.

There have been numerous studies analyzing the relationship between test suite size, code coverage, and test suite effectiveness [13, 14, 16, 17, 18, 19, 22]. More recently, Inozemtseva and Holmes [20] found that there is a moderate to very strong correlation between the effectiveness of a test suite and the number of test methods, but only a low to moderate correlation between the effectiveness and code coverage when the test suite size is controlled for. These findings imply that (1) the more test cases there are, the more effective a test suite becomes, (2) the more test cases there are, the higher the coverage, and thus (3) test suite size plays a prominent role in the observed correlation between coverage and effectiveness.

All these studies treat test methods as the smallest unit of interest. However, we believe such coarse-grained studies are not sufficient to show the main factors influencing a test suite's effectiveness. In this paper, we propose to dissect test methods and investigate why test suite size correlates strongly with effectiveness. To that end, we focus on test assertions inside test methods. Test assertions are statements in test methods through which desired specifications are checked against actual program behaviour. As such, assertions are at the core of test methods.

We hypothesize that assertions[1] have a strong influence on test suite effectiveness, and this influence, in turn, is the underlying reason behind the strong correlation between test suite size, code coverage, and test suite effectiveness. To the best of our knowledge, we are the first to conduct a large-scale empirical study on the direct relationship between assertions and test suite effectiveness.

In this paper, we conduct a series of experiments to quantitatively study the relationship between test suite effectiveness and the (1) number of assertions, (2) assertion coverage, and (3) different types of assertions.

This paper makes the following main contributions:

- The first large-scale study analyzing the relation between test assertions and test suite effectiveness. Our study composes 6,700 test suites in total, from 5,892 test cases and 24,701 assertions of five real-world Java projects in different sizes and domains.

[1]We use the terms 'assertion' and 'test assertion' interchangeably in this paper.

- Empirical evidence that (1) test assertion quantity and assertion coverage are strongly correlated with a test suite's effectiveness, (2) assertion quantity can significantly influence the relationship between a test suite's size and its effectiveness, (3) the correlation between statement coverage and effectiveness decreases dramatically when assertion coverage is controlled for.
- A classification and analysis of the effectiveness of assertions based on their properties, such as (1) creation strategy (human-written versus automatically generated), (2) the content type asserted on, and (3) the actual assertion method types.

2. RELATED WORK

Code coverage is measured as the percentage of program code that is executed at least once during the execution of the test suite. There is a large body of empirical studies investigating the relationship between different coverage metrics (such as statement, branch, MC/DC) and test suite effectiveness [13, 14, 16, 17, 18, 19]. All these studies find some degree of correlation between coverage and effectiveness. However, coverage remains a controversial topic [14, 20] as there is no strong evidence for its direct relation with effectiveness. The reason is that coverage is necessary but not sufficient for a test suite to be effective. For instance, a test suite might achieve 100% coverage but be void of test assertions to actually check against the expected behaviour, and thus be ineffective.

Researchers have also studied the relationship between test suite size, coverage, and effectiveness [22, 20]. In these papers, test suite size is measured in terms of the number of test methods in the suite. Different test suites, with size controlled, are generated for a subject program to study the correlation between coverage and effectiveness. Namin and Andrews [22] report that both size and coverage independently influence test suite effectiveness. More recently, Inozemtseva and Holmes [20] find that size is strongly correlated with effectiveness, but only a low to moderate correlation exists between coverage and effectiveness when size is controlled for. None of these studies, however, looks deeper into the test cases to understand why size has a profound impact on effectiveness. In our work, we investigate the role test assertions play in effectiveness.

Schuler and Zeller [24] propose the notion of 'checked coverage'[2] as a metric to assess test oracle quality. Inspired by this work, we measure assertion coverage of a test suite as the percentage of statements directly covered by the assertions. We are interested in assertion coverage because it is a metric directly related with the assertions in the test suite. In the original paper [24], the authors evaluated the metric by showing that there is a similar trend between checked coverage, statement coverage, and mutation score. In this paper, we conduct an empirical study on the correlation level between assertion coverage and test suite effectiveness. In addition, we compose a large set of test suites (up to thousands) for each subject under test, whereas only seven test suites were compared in the original paper. Moreover, we study the correlation between statement coverage and test suite effectiveness, to compare with the relationship between assertion coverage and test suite effectiveness, by composing test suites with assertion coverage controlled.

Cai and Lyu [14] studied the relationship between code coverage and fault detection capability under different testing profiles. They found that the effect of code coverage on fault detection varies under different testing profiles. Also, the correlation between the two measures is strong with exceptional test cases, while weak in normal testing settings. However, they did not examine the role assertions might play in different profiles on the effectiveness of test cases. To the best of our knowledge, we are the first to investigate the influence of different assertion properties on suite effectiveness. We classify assertion properties in three categories, and study the effectiveness of each classification separately.

[2]We use the terms 'assertion coverage' and 'checked coverage' interchangeably in this paper.

3. EXPERIMENTAL DESIGN

The goal of this paper is to study the relationship between assertions and test suite effectiveness. To achieve this goal, we design controlled experiments to answer the following research questions:

RQ1 Is the number of assertions in a test suite correlated with effectiveness?

RQ2 Is the assertion coverage of a test suite correlated with effectiveness?

RQ3 Does the type of assertions in a test suite influence effectiveness?

We examine these three main aspects of assertions in our study because (1) almost all test cases contain assertions, but the number of assertions varies across test suites (see Table 2); we aim to investigate if the number of assertions plays a role in effectiveness, (2) the fraction of statements in the source code executed and checked directly by assertions should intuitively be closely related to effectiveness; we set out to explore if and to what degree this is true; and (3) assertions have different characteristics, which may potentially influence a test suite's effectiveness, such as their method of creation (e.g., human-written, automatically generated), the type of arguments they assert on (e.g., boolean, string, integer, object), and the assertion method itself (e.g., `assertTrue`, `assertEquals`).

All our experimental data is publicly available.[3]

3.1 Terminology

Test case: a JUnit4 test method annotated with `@Test`. We use the terms 'test method' and 'test case' interchangeably in this paper.

Test suite: the collection of a subject program's test cases.

Test suite size: number of test cases in a test suite.

Master/original test suite: the test suite written by the developers of a subject program.

3.2 Subject Programs

To automate data collection, we selected Java programs that use Apache Maven[4] as their build system, and JUnit4 as their testing framework. We select programs of different sizes to ensure the experiment results are not project size dependent.

Our set of subjects contains five projects in different application domains. JFreeChart [6] is a free Java chart library for producing charts. Apache Commons Lang [1] is a package

[3]`http://salt.ece.ubc.ca/software/assertion-study/`

[4]`http://maven.apache.org`

Table 1: Characteristics of the subject programs.

ID	Subjects	Java SLOC	Test SLOC	Test cases	Assertions	Statement coverage	Assertion coverage
1	JFreeChart [6]	168,777	41,382	2,248	9,177	45%	30%
2	Apache Commons Lang [1]	69,742	41,301	2,614	13,099	92%	59%
3	Urban Airship Java Library [10]	35,105	11,516	503	1,051	72%	53%
4	lambdaj [7]	19,446	4,872	310	741	93%	65%
5	Asterisk-Java [2]	36,530	4,243	217	633	24%	10%
Total/Average		329,600	103,314	5,892	24,701	45%	30%

Table 2: Number of assertions per test case.

ID	Min	1st Q.	Median	3rd Q.	Max	Mean	σ
1	0	1	2	4	114	4.1	6.7
2	0	1	3	6	104	5.1	7.4
3	0	0	1	2	42	2.1	3.6
4	0	1	2	3	21	2.8	3.2
5	0	1	2	3	17	3.0	2.9

of Java utility classes for the classes that are in java.lang's hierarchy. Urban Airship Java Library [10] is a Java client library for the Urban Airship API. Lambdaj [7] is a Java project for manipulating collections in a pseudo-functional and statically typed way. The last subject, Asterisk-Java [2], is a free Java library for Asterisk PBX integration.

The characteristics of these subject programs are summarized in Table 1. Lines of source code are measured using SLOCCount [9]. Columns 5–8 illustrate test suite size in terms of number of test methods, assertion quantity, statement coverage, and assertion coverage, of each subject's master test suite, respectively. Table 2 presents descriptive statistics regarding the number of assertions per test case for the subject systems.[5]

3.3 Procedure

To study the relationship between assertions and test suite effectiveness, a large set of test suites with different assertion related properties are required. In this section, we present how the experiments are conducted with respect to each research question. We first discuss the variables of interest, then explain how test data are collected by generating new test suites, and finally describe how the results are analyzed.

3.3.1 Effectiveness of Assertion Quantity (RQ1)

In order to answer RQ1, we investigate three variables, namely, number of test methods, number of assertions, and test suite effectiveness. We collect data by generating test suites in three ways, (1) randomly, (2) controlling test suite size, and (3) controlling assertion quantity. For each set of test suites, we compute the correlation between the three variables.

Number of test cases. We implemented a tool that uses the JavaParser [4] library to identify and count the total number of test cases in a given test suite.

Number of assertions. For each identified test case, the tool counts the number of test assertions (e.g., `assertTrue`) inside the body of the test case.

Test suite effectiveness. Effectiveness captures the fault detection ability of a test suite, which can be measured as a percentage of faults detectable by a test suite. To measure the fault detection ability of a test suite, a large number of known real faults are required for each subject, which is practically unachievable. Instead, researchers generate artificial faults that resemble developer faults using techniques such as mutation testing. In mutation testing, small syntactical changes are made to random locations in the original program to generate a large number of mutants. The test suite is then run against each mutant. A mutant is killed if any of the test case assertions fail or the program crashes.

Mutation score. The mutation score, calculated as a percentage of killed mutants over total number of non-equivalent mutants, is used to estimate fault detection ability of a test suite. Equivalent mutants are syntactically different but semantically the same as the origin program, and thus undetectable by any test case. Since there is no trivial way of identifying equivalent mutants, similar to other studies [20], we treat all mutants that cannot be detected by a program's original (master) test suite, as equivalent mutants when calculating mutation scores for our generated test suites.

Mutations are produced by transforming a program syntactically through mutation operators, and one could argue about the eligibility of using the mutation score to estimate a test suite's effectiveness. However, mutation testing is extensively used as a replacement of real fault detection ability in the literature [14, 20, 22]. There is also empirical evidence confirming the validity of mutation testing in estimating test suite effectiveness [11, 12, 15, 21].

We use the open source tool PIT [8] to generate mutations. We tested each of our subject programs to ensure their test suites can successfully execute against PIT. We use PIT's default mutation operators in all of our experiments.

Generating test suites. To answer RQ1, we generate test suites in three different ways, from the master test suites of the subject programs.

Random test suites. We first generate a set of test suites by randomly selecting a subset of the test cases in the master test suite, without replacement. The size of each generated test suite is also randomly decided. In other words, we generate this set of test suites without controlling on test suite size or assertion quantity.

Controlling the number of test methods. Each test case typically has one or more assertions. A test suite with more test cases is likely to contain more assertions, and vice versa. From our observations, if test suites are randomly generated, there exists a linear relationship between test suite size and the number of assertions in the suites. If there is a linear relationship between two properties A (e.g., assertion quantity) and B (e.g., suite size), a relationship between A and a third property C (e.g., effectiveness) can easily transform to a similar relationship between B and C through transitive closure. To remove such indirect influences, we generate a second set of test suites by controlling the size. More specifically, a target test suite contains all of the

[5] We were surprised to see such high max numbers of assertions per test case, so we manually verified these numbers. For instance, the 114 max assertions for JFreeChart are in the `testEquals` test method of the `org.jfree.chart.plot.CategoryPlotTest` class.

test methods but only a subset of the assertions from the master test suite. Based on the total number of assertions in the master test suite, we first select a base number b, which indicates the size of the smallest test suite, and a step number x, which indicates size differences between test suites. Therefore, the i-th test suite to be generated, contains all of the test cases but only $b + x * i$ randomly selected assertions of the master test suite.

Controlling the number of assertions. We also generate another set of test suites by controlling on assertion quantity. To achieve this, we first assign test cases to disjoint *buckets* according to the number of assertions they contain. For instance, for JFreeChart, test cases are assigned to three disjoint buckets, where bucket *low* contains test cases with 2 or less assertions, bucket *middle* contains test cases with 3 or 4 assertions, and bucket *high* contains the rest of test cases which have 5 or more assertions. We divide test cases in this way such that each bucket has a comparable size. Then we generate 100 test suites from each of the buckets randomly without replacement. Following this process, with a similar test suite size, test suites generated from bucket *high* always contain more assertions than test suites generated from bucket middle, and so forth.

Correlation analysis. For RQ1, we use Pearson and Kendall's correlation to quantitively study the relationship between test suite size, assertion quantity, and test suite effectiveness. The Pearson correlation coefficient indicates the strength of a linear relationship between two variables. The Kendall's correlation coefficient measures the extent to which, as one variable increases, the other variable tends to increase, without requiring that increase to be represented by a linear relationship.

3.3.2 Effectiveness of Assertion Coverage (RQ2)

To answer RQ2, we measure a test suite's assertion coverage, statement coverage, and effectiveness. We collect data by first looking at the set of test suites which were randomly generated for RQ1, then generate a new set of test suites by controlling their assertion coverage. For each of the two sets of test suites, we study and compare the correlations between the three variables using the same analysis methods as described in Section 3.3.1.

Explicit mutation score. Not all detectable faults in a program are detected by test assertions. From our observations, mutants can either be *explicitly killed* by assertions or *implicitly killed* by program crashes. Programs may crash due to unexpected exceptions. Program crashes are much easier to detect as they do not require dedicated assertions in test cases. On the other hand, all the other types of faults that do not cause an obvious program crash, are much more subtle and require proper test assertions for their detection. Since the focus of our study is on the role of assertions in effectiveness, in addition to the mutation score, we also compute the *explicit mutation score*, which measures the fraction of mutants that are explicitly killed by the assertions in a test suite. Table 3 provides mutation data in terms of the number of mutations generated for each subject, number of mutants killed by the test suites, number of mutants killed only by test assertions (e.g., excluding crashes), and the percentage of mutants killed by assertions with respect to the total number of killed assertions.

From what we have observed in our experiments, PIT always generates the same set of mutants for a piece of source code when executed multiple times. Thus, to measure the *explicit mutation score* of a test suite, we remove all assertions from the test suite, measure its mutation score again, and then subtract the fraction of implicit killed mutants from the original mutation score.

Table 3: Mutation data for the subject programs.

ID	Mutants	Killed (#)	Killed by Assertions (#)	Killed by Assertions (%)
1	34,635	11,299	7,510	66%
2	11,632	9,952	7,271	73%
3	4,638	2,546	701	28%
4	1,340	1,084	377	35%
5	4,775	957	625	65%

Assertion coverage. Assertion coverage, also called checked coverage [24], measures the fraction of statements in the source code executed via the backward slice of the assertion statements in a test suite.

We use the open source tool JavaSlicer [5] to identify *assertion checked statements*, which are statements in the source code executed through the execution of assertions in a test suite. JavaSlicer is an open-source dynamic slicing tool, which can be used to produce traces of program executions and offline dynamic backward slices of the traces. We automatically identify checked statements of a test suite by (1) identifying all assertion statements and constructing slicing criteria, (2) using JavaSlicer to trace each test class separately, and (3) mining the traces computed in the previous step to identify dynamic backward slices of the assertions, and finally (4) since each backward slice of an assertion includes statements from the test case, calls to the JUnit APIs, and statements from the source code, we filter out the data to keep only the statements pertaining to the source code.

For large test suites, we observed that using JavaSlicer is very time consuming. Thus, we employ a method to speed up the process in our experiments. For all the test classes in each original master test suite, we repeat steps 1–4 above, to compute the *checked statements* for each test method individually. Each statement in the source code is uniquely identified by its classname and line number and assigned an ID. We then save information regarding the *checked statements* of each test method into a data repository. Once a new test suite is composed, its *checked statements* can be easily found by first identifying each test method in the test suite, then pulling the *checked statements* of the test method from the data repository, and finally taking a union of the *checked statements*. The *assertion coverage* of a generated test suite is thus calculated as the total number of *checked statements* of the suite divided by the total number of statements.

Statement coverage. Unlike assertion coverage, which only covers what assertion statements execute, statement coverage measures the fraction of the source code covered through the execution of the whole test suite.

In this paper, we select statement coverage out of the traditional code coverage metrics as a baseline to compare with assertion coverage. The reason behind our selection is twofold. First, statement coverage is one of the most frequently used code coverage metrics in practice since it is relatively easy to compute and has proper tool support. Second, two recent empirical studies suggest that statement coverage is at least as good as any other code coverage metric in predicting effectiveness. Gopinath et al. [19] found

statement coverage predicts effectiveness best compared to block, branch, or path coverage. Meanwhile, Inozemtseva and Holmes [20] found that stronger forms of code coverage (such as decision coverage or modified condition coverage) do not provide greater insights into the effectiveness of the suite. We use Clover [3], a highly reliable industrial code coverage tool, to measure statement coverage.

Generating test suites. To answer RQ2, we first use the same set of test suites that were randomly generated for RQ1. In addition, we compose another set of test suites by controlling assertion coverage. We achieve this by controlling on the number of *checked statements* in a test suite. Similarly, based on the total number of checked statements in the master test suite of a program, we predefine a base number b, which indicates assertion coverage of the smallest test suite, and a step number x, which indicates assertion coverage differences between the test suites. When generating a new test suite, a hash set of the current checked statements is maintained. If the target number for checked statements is not reached, a non-duplicate test method will be randomly selected and added to the test suite. To avoid too many trials of random selection, this process is repeated until the test suite has $[b + x * i, (b + x * i) + 10]$ checked statements. This way, the i-th target test suite has an assertion coverage of $(b + x * i)/N$, where N is the total number of statements in the source code.

3.3.3 Effectiveness of Assertion Types (RQ3)

RQ3 explores the effectiveness of different characteristics of test assertions. To answer this research question, we first automatically assign assertions to different categories according to their characteristics, then generate a set of sample test suites for each category of assertions, and finally conduct statistical analysis on the data collected from the generated test suites.

Assertion categorization. Assertions can be classified according to their characteristics. Some of these characteristics may be potential influence factors to test suite effectiveness. We categorize assertions in three ways:

Human-written versus generated. Human-written test cases contain precise assertions written by developers about the expected program behaviour. On the other hand, automatically generated tests contain generic assertions. Commonly, it is believed that human-written test cases have a higher fault detection ability than generated assertions. We test this assumption in our work.

Assertion content type. Test assertions either check the value of primitive data type or objects of different classes. We further classify Java's primitive data types into *numbers* (for int, byte, short, long, double, and float), *strings* (for char and String), and *booleans*. This way, depending on the type of the content of an assertion, it falls into one of the following classes: *number-content-type*, *string-content-type*, *boolean-content-type*, or *object-content-type*. We explore whether these assertion content types have an impact on the effectiveness of a test suite.

For assertion content type, we apply dynamic analysis to automatically classify the assertions in a given test suite to the different categories. We first instrument test code to probe each assert statement for the type of content it asserts on. Then, we run the instrumented test code, and use the information collected to automatically assign assertions to the different content type categories.

Assertion method type. It is also possible to categorize assertions according to their actual method types. For instance, `assertTrue` and `assertFalse`, `assertEquals` and `assertNotEquals`, and `assertNull` and `assertNotNull` can be assigned to different categories. We investigate if these assertion method types have an impact on effectiveness.

For assertion method types, we parse the test code and syntactically identify and classify assertions to different assertion method type classes.

Generating test suites. Under each assertion categorization, for each assertion type, we compose 50 sample test suites, each containing 100 assertions. A sample test suite contains all test methods in the master test suite, but only 100 randomly selected assertions of the target type. For instance, a sample test suite of the type *string-content-type* will contain all the test methods in the master test suite but only 100 randomly selected *string-content-type* assertions.

To quantitively compare the effectiveness between human-written and generated assertions, for each subject program, we generate (1) 50 sample test suites, each containing 100 human-written assertions from the master test suite, and (2) 50 sample test suites, each containing 100 automatically generated assertions using Randoop [23], a well-known feedback-directed test case generator for Java. We use the default settings of Randoop.

Analysis of variances. For assertion content type and assertion method type, since there are multiple variables involved, we use the One-Way ANOVA (analysis of variance) statistical method to test whether there is a significant difference in test suite effectiveness between the variables. Before we conduct the ANOVA test, we used the Shapiro-Wilk test to pretest the normality of our data, and Levene's test to pretest the homogeneity of their variances. Both were positive. ANOVA answers the question whether there are significant differences in the population means. However, it does not provide any information about how they differ. Therefore, we also conduct a Tukey's Honest Significance Test to compare and rank the effectiveness of assertion types.

4. RESULTS

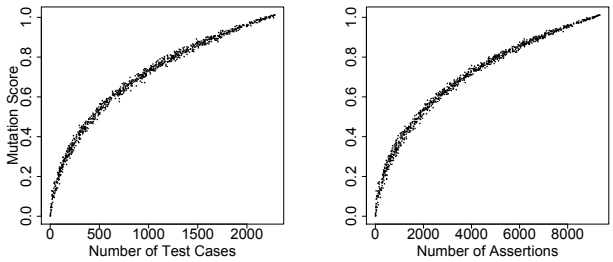
In this section, we present the results of our experiments.

4.1 Effectiveness of Assertion Quantity

Ignoring test suite size. Figure 1 depicts plots of our collected data for JFreeChart.[6] Figures 1a and 1b show that the relationship between test suite size and effectiveness is very similar to the relationship between assertion quantity and effectiveness. As the plot in Figure 1c shows, there exists a linear relationship between the number of test methods and the number of assertions, in the 1000 randomly generated test suites.

Table 4 shows the Pearson (ρ_p) and Kendall's (ρ_k) correlations between effectiveness with respect to suite size (m) and assertion quantity a, for the test suites that are randomly generated for all the five subjects. As the table shows, there is a very strong correlation between number of assertions in a test suite and the test suite's effectiveness, and the correlation coefficients are very close to that of suite size and effectiveness. This is consistent with the plots of Figure 1.

[6]Note that we observed a similar trend from the other subjects, and only include plots for JFreeChart due to space limitations.



(a) (b)

(c)

Figure 1: Plots of (a) suite size versus effectiveness, (b) assertion quantity versus effectiveness, and (c) suite size versus assertion quantity, for the 1000 randomly generated test suites from JFreeChart. The other four projects share a similar pattern consistently.

Table 4: Correlation coefficients between test suite size and effectiveness (m), and assertion quantity and effectiveness (a). ρ_p shows Pearson correlations and ρ_k represents Kendall's correlations.

Subject ID	$\rho_p(m)$	$\rho_p(a)$	$\rho_k(m)$	$\rho_k(a)$	p-value
1	0.954	0.954	0.967	0.970	
2	0.973	0.973	0.969	0.969	
3	0.927	0.927	0.917	0.917	$< 2.2e-16$
4	0.929	0.928	0.912	0.930	
5	0.945	0.947	0.889	0.894	

The correlations between assertion quantity and effectiveness are slightly higher or equal to the correlations between the number of test methods and effectiveness.

***Finding 1**: Our results indicate that, without controlling for test suite size, there is a very strong correlation between the effectiveness of a test suite and the number of assertions it contains.*

Controlling for test suite size. Table 5 shows our results when we control for test suite size. Column 2 shows the number of assertions in the smallest test suite, and column 3 shows the difference in assertion quantity between generated test suites. Columns 3 and 4 present the Pearson and Kendall's correlations, respectively, between the assertion quantity and the effectiveness of the test suites that are generated from the five subjects by controlling test suite size. As the high correlation coefficients indicate in this table, even when test suite size is controlled for, there is a very strong correlation between effectiveness and the number of assertions.

Table 5: Correlations between number of assertions and suite effectiveness, when suite size is controlled for.

Subject ID	Base	Step	$\rho_p(a)$	$\rho_k(a)$	p-value
1	1,000	50	0.976	0.961	
2	100	100	0.929	0.970	
3	0	10	0.948	0.846	$< 2.2e-16$
4	100	10	0.962	0.839	
5	100	5	0.928	0.781	

***Finding 2**: Our results suggest that, there is a very strong correlation between the effectiveness of a test suite and the number of assertions it contains, when the influence of test suite size is controlled for.*

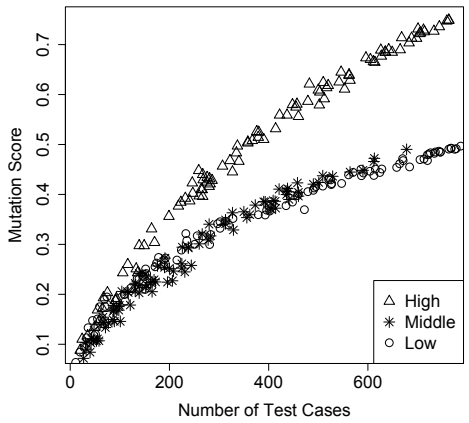


Figure 2: Plot of mutation score against suite size for test suites generated from assertion buckets *low*, *middle*, and bucket *high* from JFreeChart. The other four projects share a similar pattern consistently.

Controlling for assertion quantity. Figure 2 plots the effectiveness of the test suites generated by controlling for the number of assertions. Three buckets of high, middle and low in terms of the number of assertions were used for generating these test suites (see Section 3.3.1). From low to high, each bucket contained 762, 719, and 742 test cases in total, and the average number of assertions per test case was 0.9, 2.5, and 9.1, respectively. From the curves in the plot, we can see that the effectiveness increases as the number of test methods increase. However, comparing the curves, there is a clear upward trend in a test suite's effectiveness as its assertion quantity level increases. For every given test suite taken from the lower curve on the plot, there exits a test suite with the same suite size that has a higher effectiveness because it contains more assertions.

***Finding 3**: Our results indicate that, for the same test suite size, assertion quantity can significantly influence the effectiveness.*

4.2 Effectiveness of Assertion Coverage

To answer RQ2, we first computed the assertion coverage, statement coverage, and mutation score of the randomly

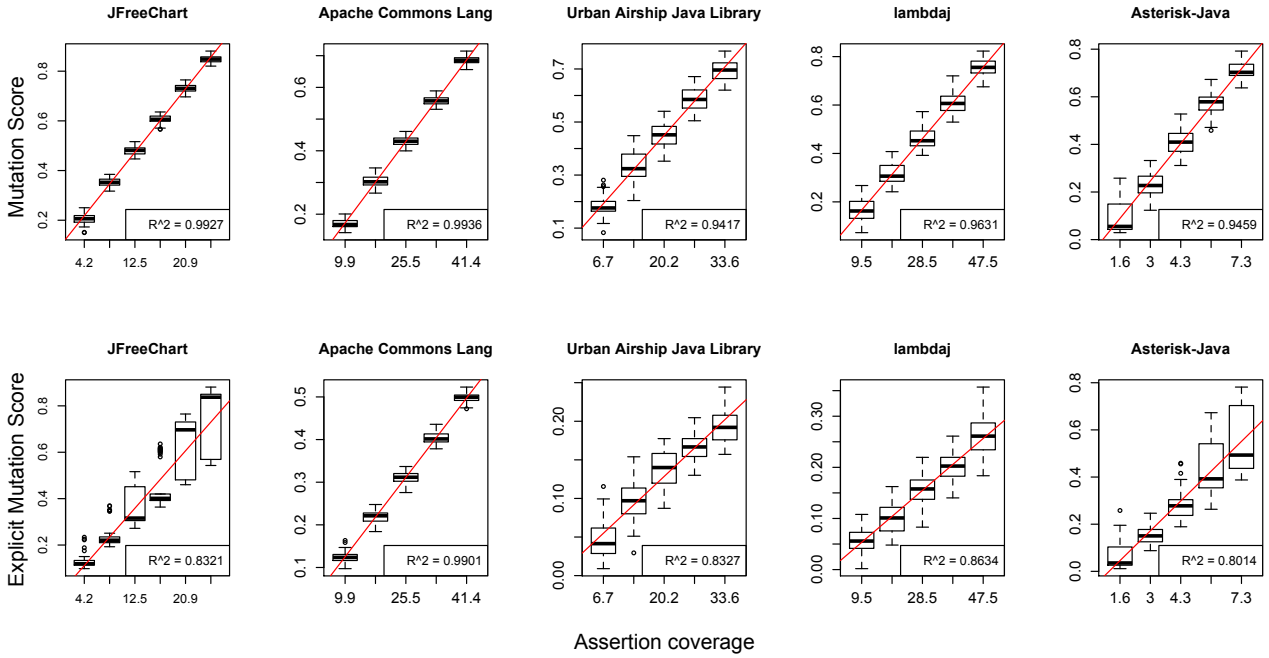
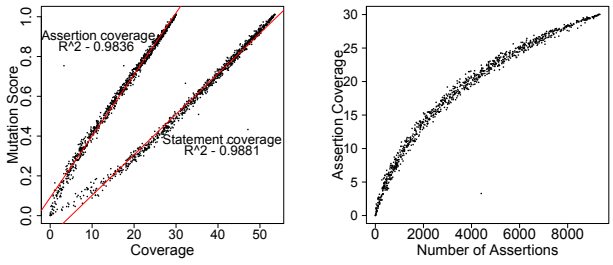


Figure 3: Mutation score (above) and explicit mutation score (below) plot against assertion coverage for the five subject programs. Each box represents the 50 test suites of a given assertion coverage that were generated from the original (master) test suite for each subject. The Kendall's correlation is 0.88–0.91 between assertion coverage and mutation score, and 0.80–0.90 between assertion coverage and explicit mutation score.

generated test suites (see subsubsection 3.3.2). Figure 4a plots our results. The two fitted lines both have a very high adjusted R^2 and p-value smaller than $2.2e-16$; this indicates a very strong correlation between assertion coverage and effectiveness as well as statement coverage and effectiveness. The plot also shows that a test suite having the same assertion coverage as another test suites's statement coverage, is much more effective in detecting faults. Compared with statement coverage, assertion coverage is a more sensitive predictor of test suite effectiveness.



(a) **(b)**

Figure 4: Plots of (a) mutation score against assertion coverage and statement coverage, (b) assertion coverage against assertion quantity, for the 1000 randomly generated test suites from JFreeChart.

Figure 4b plots assertion coverage against number of assertions in a test suite. From the plot, assertion coverage of a test suite increases as test suite size increases. However, the increasing rate of assertion coverage decreases as test suite size increases. There is a strong increasing linear relationship between assertion coverage and test suite effectiveness. Therefore, it is expected that, test suite effectiveness increases as test suite size increases but with a diminishing increasing rate, which is again consistent with our results in subsection 4.1.

> ***Finding 4****: Our results suggest that, assertion coverage is very strongly correlated with test suite effectiveness. Also, ignoring the influence of assertion coverage, there is a strong correlation between statement coverage and the effectiveness.*

Controlling for assertion coverage. Figure 3 shows box plots of our results for the test suites generated by controlling their assertion coverage. The adjusted R^2 value for each regression line is shown in the bottom right corner of each box plot. It ranges from 0.94 to 0.99 between assertion coverage and mutation score, and 0.80 to 0.99 between assertion coverage and explicit mutation score. This indicates assertion coverage can predict both mutation score and explicit mutation score well.

Table 6 summarizes statistics for these test suites. Column 3 contains the Kendall's correlations between statement coverage and mutation score (0.50–0.76), column 4 presents the Kendall's correlations between statement coverage and explicit mutation score (0.01–0.63). When assertion coverage is controlled for, there is a *moderate to strong* correlation between statement coverage and mutation score, and only a *low to moderate* correlation between statement coverage and explicit mutation score. For instance, only about 1/3 of the mutants generated for Urban Airship Library (ID 5) and lambdaj (ID 4) are explicitly detectable mutants;

Table 6: Statistics of test suites composed at different assertion coverage levels.

Subject ID	Assertion Coverage	Stat. Coverage Corr.		Mutation Score		Statement Coverage
		$\rho_{general}$	$\rho_{explicit}$	*general*	*explicit*	
1	4.2%	0.62	0.17	0.21	0.13	15%
	8.4%	0.60	0.22	0.35	0.24	23%
	12.5%	0.59	0.11	0.48	0.35	30%
	16.7%	0.63	0.33	0.61	0.45	36%
	20.9%	0.58	0.26	0.73	0.61	41%
	25.1%	0.71	0.32	0.85	0.75	47%
2	9.9%	0.67	0.49	0.17	0.12	21%
	17.7%	0.67	0.51	0.30	0.22	34%
	25.5%	0.65	0.48	0.43	0.31	46%
	33.7%	0.66	0.50	0.56	0.40	57%
	41.4%	0.58	0.27	0.69	0.50	68%
3	6.7%	0.62	0.06	0.18	0.05	19%
	13.5%	0.74	0.06	0.33	0.10	30%
	20.2%	0.76	0.01	0.46	0.14	39%
	26.9%	0.75	0.07	0.59	0.17	48%
	33.6%	0.76	0.05	0.70	0.20	55%
4	9.5%	0.76	0.21	0.17	0.06	19%
	19.0%	0.73	0.33	0.31	0.10	32%
	28.5%	0.70	0.30	0.46	0.15	45%
	38.0%	0.63	0.23	0.61	0.20	58%
	47.5%	0.50	0.10	0.76	0.26	70%
5	1.6%	0.73	0.63	0.10	0.06	4%
	3.0%	0.76	0.35	0.23	0.15	8%
	4.3%	0.70	0.38	0.41	0.28	12%
	5.8%	0.60	0.25	0.57	0.43	16%
	7.3%	0.62	0.24	0.71	0.56	19%

correspondingly there is only a weak correlation (0.01–0.33) between their statement coverage and explicit mutation score. A higher fraction ($\approx 2/3$) of the mutants generated for the other three subjects are explicitly detectable mutants, and thus the correlation between their statement coverage and explicit mutation score increases significantly (from 0.11 to 0.63).

Columns 5–7 in Table 6 pertain to the average mutation score, average explicit mutation score, and average statement coverage of the test suites at each assertion coverage level, respectively. As the results show, a slight increase in assertion coverage can lead to an obvious increase in the mutation score and explicit mutation score. For instance, for JFreeChart (ID 1), when assertion coverage increases by around 4%, the mutation score increases by around 12.4% and explicit mutation score increases by around 11%. On the other hand, a 4% increase in the statement coverage does not always increase either mutation score or explicit mutation score. This shows again that assertion coverage is a more sensitive indicator of test suite effectiveness, compared to statement coverage.

***Finding 5**: Our results suggest that, assertion coverage is capable of predicting both mutation score and explicit mutation score. With assertion coverage controlled for, there is only a moderate to strong correlation between statement coverage and mutation score, and a low to moderate correlation between statement coverage and explicit mutation score. Test suite effectiveness is more sensitive to assertion coverage than statement coverage.*

4.3 Effectiveness of Assertion Types

To answer RQ3, we examined the 9,177 assertions of JFreeChart.

Assertion generation strategy. Figure 5a plots the effectiveness of human-written test suites and Randoop generated test suites against assertion quantity. As we can observe, the effectiveness of human-written and generated test suites both increase as the assertion quantity increases. However, the effectiveness of the generated test suites gets saturated much faster than human-written test suites.

From our observations of the composed test suites, the 50 human-written sample test suites are effective in killing mutants, while the 50 generated test suites can hardly detect any mutant. We increased the assertion quantity in the sample test suites to 500, but still saw the same pattern.

***Finding 6**: Our results indicate that, human-written test assertions are far more effective than automatically generated test assertions.*

Assertion content type. Assertions are also classified based on the types of the content they assert on. Figure 5b box plots the effectiveness of the sample test suites that exclusively contain assertions on object, boolean, number, or string types. Tables 7 and 8 show the ANOVA and the Tukey's Honest Significance test, respectively. The F value is 1544 with a p-value very close to 0, thus we can confidently reject the null hypothesis of equal variances (effectiveness) for the four assertion content types. Table 8 shows the estimated difference in mutation score in column 2, and the 95% confidence interval of the difference in columns 3 and 4. The Tukey's test indicates that there is a significant difference between the effectiveness of assertions that assert on boolean/object, string, and number types. Assertions that assert on boolean types are as effective as assertions that assert on objects.

***Finding 7**: Our results suggest that, there is a significant difference between the effectiveness of assertions that assert on different content types. Their effectiveness can be ranked in increasing order: (1) string, (2) number, (3) boolean or object type.*

Assertion method type. Assertions can also be classified by their actual method types. Figure 5c plots the effectiveness

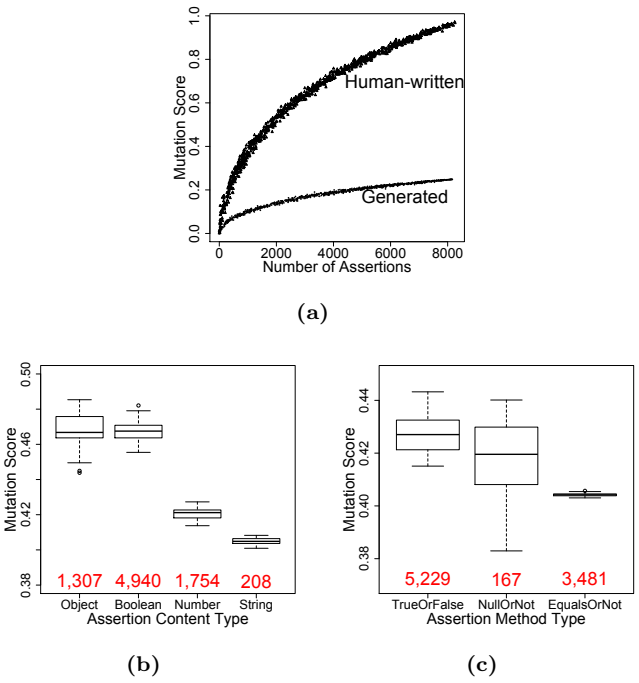


Figure 5: Plots of (a) assertion quantity versus effectiveness of human-written and generated tests, (b) assertion content types versus effectiveness, and (c) assertion method types versus effectiveness. In (b) and (c), each box represents the 50 sample test suites generated for each type; the total number of assertions of each type are indicated in red.

Table 7: One-Way ANOVA on the effectiveness of assertion content types and actual assertion types.

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
Assertion Content Types					
Type	3	0.15675	0.05225	1544	<2e-16
Residuals	196	0.00663	0.00003		
Assertion Method Types					
Type	2	0.01398	0.006988	87.87	<2e-16
Residuals	147	0.01169	0.000080		

of the sample test suites that belong to the three assertion method types. In this paper, we did not study *assertSame* and *assertNotSame*, because there were only 27 of them in JFreeChart, which is a low number to be representative. The bottom half of tables 7 and 8, present the ANOVA and Tukey's Honest Significance test, respectively, for assertion method types. The F value is 87.87 with a p-value very close to 0, thus we can reject the null hypothesis of equal variances (effectiveness) for the three assertion method types. The Tukey's test shows that there exists a significant difference between the effectiveness of the three assertion method types.

***Finding 8**: Our results indicate that, there is a significant difference between the effectiveness of different assertion method types. The effectiveness of the method types can be ranked from low to high as: (1) `assert(Not)Null`, (2) `assert(Not)Equals`, and (3) `assertTrue/False`.*

Table 8: Tukey's Honest Significance Test on the effectiveness of assertion content types and assertion method types. Each of the sample test suites used for the comparison contains 100 assertions of a target type.

Types	diff	lwr	upr	p adj
Assertion Content Types				
Boolean vs. Object	-0.0002	-0.0032	0.0028	0.9985
Number vs. Boolean	-0.0470	-0.0500	-0.0440	0.0000
String vs. Number	-0.0156	-0.0186	-0.0126	0.0000
Assertion Method Types				
assertNull/Not vs. assertTrue/False	-0.0103	-0.0145	-0.0060	1e-07
assertEquals/Not vs. assertNull/Not	-0.0133	-0.0175	-0.0091	0e+00

5. DISCUSSION

5.1 Test Suite Size vs. Assertion Quantity

From the findings 1 and 2, the number of assertions in a test suite is very strongly correlated with its effectiveness with or without controlling for the influence of test size. However, according to finding 3, if we in turn control for the number of assertions, a test suite's effectiveness at a same test size level can be directly influenced by the number of assertions it contains. Thus, test suite size is not sufficient in predicting the effectiveness without considering the influence of assertion quantity. In addition, assertion quantity provides extra indications about the suite's explicit mutation score, which constitutes a large portion of the mutation score. Therefore, test suite size can predict the effectiveness only under the assumption that there is a linear relationship between the number of test methods and the number of assertions in the test suite. We believe this is an interesting finding, which explains why previous studies [20] have found a strong correlation between suite size and effectiveness.

5.2 Implicit vs. Explicit Mutation Score

We noticed an interesting phenomenon, namely, that mutants that are implicitly detectable can also be detected by assertions, if the mutated statement falls in the coverage of the assertion. However, mutants that are explicitly detectable by assertions can never be detected by non-assertion statements of the tests. This is because explicitly detectable mutants cannot be detected by simply executing the mutated part of a program; i.e., a specific assertion statement is required to catch the program's unexpected behaviour. This is due to the fact that explicitly detectable mutants inject logical faults into a program that lead to a contradiction with the programmers' expectations. From our observations, more than half of all detectable mutants (28%–73%) are explicitly detected by assertions in a test suite; and therefore assertions strongly influence test suite effectiveness. If we only focus on explicitly detectable mutants, then test assertions are the only means to achieve suite effectiveness. This might also explain why statement coverage achieves a relatively low correlation with explicit mutation score.

5.3 Statement vs. Assertion Coverage

From findings 4 and 5, assertion coverage is a good estimator of both mutation score and explicit mutation score. If the influence of assertion coverage is controlled, there is a passable level of correlation between statement coverage

and mutation score, while only a weak correlation between statement coverage and explicit mutation score. Therefore, statement coverage is a valid estimator of mutation score only under the assumption that not all of generated mutants are explicitly detectable mutants. In other words, statement coverage is not an adequate metric of logical-fault detection ability. Statement coverage includes more statements than assertion coverage from source code, without providing any extra insights for predicting test suite effectiveness. Compared with statement coverage, assertion coverage is very strongly correlated with the effectiveness regardless of the distribution of implicitly or explicitly detectable mutants. Our results suggest that testers should aim at increasing the assertion coverage of their test suite instead of its statement coverage, when trying to improve a test suite's effectiveness.

5.4 Utilizing Assertion Types

Our results also confirm that generated assertions are much less effective than human-written assertions, and thus should only be used as a supplement of human-written assertions instead of a replacement. More interestingly, Findings 7 and 8 show us that assertion types can significantly influence a test suite's effectiveness. Therefore, it is indispensable to consider the distribution of assertion types in a test suite when predicting its effectiveness. This is an interesting finding that merits more empirical investigation.

5.5 Threats to Validity

Internal validity: To conduct the controlled experiments, we made use of many existing tools, such as PIT [8], Clover [3], and JavaSlicer [5]. We assumed these tools are able to produce valid results. Therefore, any erroneous behaviours of these tools might introduce unknown factors to the validity of our results. To mitigate such factors as much as possible, we tested our own code that uses these external tools. Similar to previous studies [20], we treated mutants that cannot be detected by the master test suites as equivalent mutants, which might overestimate the number of equivalent mutants. However, since we are mainly concerned with the correlations between mutation/explicit mutation score and the other metrics, subtracting a constant value from the total number of mutants generated, will not impact the correlations.

External validity: We study the relationship between assertions and test suite effectiveness using more than 24,000 assertions collected from five open source Java programs. However, programs written in Java may not be representative of the programs written in other languages. Thus, our results might not extend to other languages. Moreover, the assertions we examine in this paper are JUnit4 assertions, and our results may not apply to assertions used in other testing frameworks. We mainly looked at the 9,177 assertions for JFreeChart [6] when comparing the effectiveness of different assertion types. Although the set of assertions used is large and written by real developers, our findings may not generalize to other programs. In addition, there might be some interdependencies between assertion content types and assertion method types. For instance, it is difficult to control the confounding factor between assertions on the *boolean* content type and assertions that make use of `assertTrue/False` method type, as both will be present in each measurement; we do not know if a particular effectiveness correlation is due to the content type (boolean) or the assertion method type (`assertTrue/False`). Moreover, we used PIT to conduct mutation testing; PIT stops executing once a test assertion detects a mutant. However, it is helpful to know all the assertions that would fail when studying assertion types. We used Randoop to generate test cases with assertions, which were compared to human-written assertions. However, there also exist other test oracle generation strategies than feedback-directed random test generation, and using a different test generation strategy might influence the results. We used Randoop, because of its relative ease of use.

Our empirical data as well as the five subject programs are all available online, making our study repeatable.

6. CONCLUSIONS

In this paper, we studied the relationship between test assertions and test suite effectiveness. First, we examined the correlation between assertion quantity and the effectiveness, and further analyzed the influence of assertion quantity on the correlation between test suite size and the effectiveness. Second, we investigated the relationship between assertion coverage and suite effectiveness, and explored the impact of assertion coverage on the relation between statement coverage and effectiveness. Finally, we compared the effectiveness of different assertion characteristics. Based on an analysis of over 24,000 assertions collected from five cross-domain real-world Java programs, we found that:

- There is a very strong correlation between the *number of assertions* and test suite effectiveness, with or without controlling for the number of test methods in the test suite. Thus, the number of assertions in a test suite can significantly influence the prediction power of test suite size for the effectiveness.
- There is a very strong correlation between *assertion coverage* and test suite effectiveness. With assertion coverage controlled for, there is a moderate to strong correlation between statement coverage and mutation score, and only a weak to moderate correlation between statement coverage and explicit mutation score. Therefore, statement coverage is an adequate metric of test suite effectiveness only under the assumption that the faults to be detected are not only explicitly detectable, while assertion coverage is a good estimator of test suite effectiveness without such a constraint.
- Types of assertions can influence the effectiveness of their containing test suites: (1) human-written assertions are more effective than generated assertions, (2) assertions which assert on boolean/object content types are more effective than assertions that assert on string and number types, and (3) assertions with method type `assertTrue/False` are more effective than those with type `assertEquals/Not` and `assertNull/Not`.

Our results indicate that it might be sufficient to use the assertion quantity and assertion coverage as criteria to measure a suite's adequacy, since these two metrics are *at least* as good as suite size and statement coverage.

For future work, we would like to conduct experiments on more programs to further validate our findings. Moreover, we will conduct a qualitative analysis of a set of assertions to understand the reasons behind the effectiveness variations in different assertion types.

7. REFERENCES

[1] Apache commons lang. http://commons.apache.org/proper/commons-lang/.

[2] Asterisk-java. https://blogs.reucon.com/asterisk-java/.

[3] Clover. https://www.atlassian.com/software/clover/overview/.

[4] JavaParser. https://code.google.com/p/javaparser/.

[5] JavaSlicer. https://www.st.cs.uni-saarland.de/javaslicer/.

[6] JFreeChart. http://www.jfree.org/jfreechart/.

[7] Lambdaj. https://code.google.com/p/lambdaj/.

[8] Pit. http://pitest.org.

[9] SLOCCount. http://www.dwheeler.com/sloccount/.

[10] Urban airship java library. http://docs.urbanairship.com/reference/libraries/java/.

[11] J. H. Andrews, L. C. Briand, and Y. Labiche. Is mutation an appropriate tool for testing experiments? In *Proceedings of the International Conference on Software Engineering*, ICSE, pages 402–411. ACM, 2005.

[12] J. H. Andrews, L. C. Briand, Y. Labiche, and A. S. Namin. Using mutation analysis for assessing and comparing testing coverage criteria. *IEEE Trans. Softw. Eng.*, 32:608–624, 2006.

[13] L. Briand and D. Pfahl. Using simulation for assessing the real impact of test coverage on defect coverage. In *Proceedings of the International Symposium on Software Reliability Engineering*, ISSRE, pages 148–157. IEEE Computer Society, 1999.

[14] X. Cai and M. R. Lyu. The effect of code coverage on fault detection under different testing profiles. In *Proceedings of the International Workshop on Advances in Model-based Testing*, A-MOST, pages 1–7. ACM, 2005.

[15] M. Daran and P. Thévenod-Fosse. Software error analysis: A real case study involving real faults and mutations. In *Proceedings of the ACM SIGSOFT International Symposium on Software Testing and Analysis*, ISSTA, pages 158–171. ACM, 1996.

[16] P. G. Frankl and O. Iakounenko. Further empirical studies of test effectiveness. In *Proceedings of the 6th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE, pages 153–162. ACM, 1998.

[17] P. G. Frankl and S. N. Weiss. An experimental comparison of the effectiveness of the all-uses and all-edges adequacy criteria. In *Proceedings of the Symposium on Testing, Analysis, and Verification*, TAV4, pages 154–164. ACM, 1991.

[18] P. G. Frankl and S. N. Weiss. An experimental comparison of the effectiveness of branch testing and data flow testing. *IEEE Trans. Softw. Eng.*, 19:774–787, 1993.

[19] R. Gopinath, C. Jensen, and A. Groce. Code coverage for suite evaluation by developers. In *Proceedings of the International Conference on Software Engineering*, ICSE, pages 72–82. ACM, 2014.

[20] L. Inozemtseva and R. Holmes. Coverage is not strongly correlated with test suite effectiveness. In *Proceedings of the International Conference on Software Engineering*, ICSE, pages 435–445. ACM, 2014.

[21] R. Just, D. Jalali, L. Inozemtseva, M. D. Ernst, R. Holmes, and G. Fraser. Are mutants a valid substitute for real faults in software testing? In *Proceedings of the ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE, pages 654–665. ACM, 2014.

[22] A. S. Namin and J. H. Andrews. The influence of size and coverage on test suite effectiveness. In *Proceedings of the International Symposium on Software Testing and Analysis*, ISSTA, pages 57–68. ACM, 2009.

[23] C. Pacheco, S. K. Lahiri, M. D. Ernst, and T. Ball. Feedback-directed random test generation. In *Proceedings of the International Conference on Software Engineering (ICSE)*, pages 75–84. IEEE Computer Society, 2007.

[24] D. Schuler and A. Zeller. Assessing oracle quality with checked coverage. In *Proceedings of the International Conference on Software Testing, Verification and Validation*, ICST, pages 90–99. IEEE Computer Society, 2011.