Assessing Effectiveness of Test Suites: What Do We Know and What Should We Do?

PENG ZHANG, State Key Laboratory for Novel Software Technology, Nanjing University, China and Huawei Technologies Co., Ltd, China
YANG WANG, XUTONG LIU, ZEYU LU, YIBIAO YANG, YANHUI LI, and LIN CHEN, State Key Laboratory for Novel Software Technology, Nanjing University, China
ZIYUAN WANG, Nanjing University of Posts and Telecommunications, China
CHANG-AI SUN, University of Science and Technology Beijing, China
XIAO YU, Huawei Technologies Co., Ltd, China
YUMING ZHOU, State Key Laboratory for Novel Software Technology, Nanjing University, China

Background. Software testing is a critical activity for ensuring the quality and reliability of software systems. To evaluate the effectiveness of different test suites, researchers have developed a variety of metrics. **Problem.** However, comparing these metrics is challenging due to the lack of a standardized evaluation framework including comprehensive factors. As a result, researchers often focus on single factors (e.g., size), which finally leads to different or even contradictory conclusions. After comparing dozens of pieces of work in detail, we have found two main problems most troubling to our community: (1) researchers tend to oversimplify the description of the ground truth they use, and (2) data involving real defects is not suitable for analysis using traditional statistical indicators. **Objective.** We aim at scrutinizing the whole process of comparing test suites for our community. **Method.** To hit this aim, we propose a framework ASSENT (ev**A**luating te**S**t **S**uite **E**ffective**N**ess me**T**rics) to guide the follow-up research for evaluating a test suite effectiveness metric. ASSENT consists of three fundamental components: ground truth, benchmark test suites, and agreement indicator. Its functioning is as follows: first, users clarify the ground truth for determining the real order in effectiveness among test suites. Second, users generate a set of benchmark test suites and derive their ground truth order in effectiveness. Third, users use the metric to derive the order in effectiveness for the same test suites. Finally, users calculate the agreement indicator between the two orders derived by two metrics.

This work is partially supported by National Natural Science Foundation of China (62172205, 62072194, 62172202, 62272221, 62272037), Natural Science Foundation of Jiangsu Province (BK20231402), and the NJU-Huawei Software New Technology Joint Laboratory Fund (TC20230202021-2023-08).
Authors' addresses: P. Zhang, State Key Laboratory for Novel Software Technology, Nanjing University, 163 Xianlin Road, Qixia District, Nanjing, Jiangsu Province, China, 210023 and Huawei Technologies Co., Ltd, No. 410, Jianghong Road, Changhe Street, Binjiang District, Hangzhou, Zhejing Province, China, 310000; e-mail: dz1833034@smail.nju.edu.cn; Y. Wang, X. Liu, Z. Lu, Y. Yang (Corresponding author), Y. Li, L. Chen, and Y. Zhou (Corresponding author), State Key Laboratory for Novel Software Technology, Nanjing University, 163 Xianlin Road, Qixia District, Nanjing, Jiangsu Province, China, 210023; e-mails: {njuwy, xryu, zeyulu}@smail.nju.edu.cn, {yangyibiao, yanhuili, lchen, zhouyuming}@nju.edu.cn; Z. Wang, Nanjing University of Posts and Telecommunications, No. 9, Wenyuan Road, Yadong New District, Nanjing, Jiangsu Province, China, 210003; e-mail: wangziyuan@njupt.edu.cn; C.-ai Sun, University of Science and Technology Beijing, 30 Xueyuan Road, Haidian District, Beijing, Beijing, China, 100083; e-mail: casun@ustb.edu.cn; X. Yu, Huawei Technologies Co., Ltd, No. 410, Jianghong Road, Changhe Street, Binjiang District, Hangzhou, Zhejing Province, China, 310000; e-mail: yuxiao25@huawei.com.
Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
1049-331X/2024/04-ART86 $15.00
https://doi.org/10.1145/3635713

Result. With ASSENT, we are able to compare the accuracy of different test suite effectiveness metrics. We apply ASSENT to evaluate representative test suite effectiveness metrics, including mutation score and code coverage metrics. Our results show that, based on the real faults, mutation score, and subsuming mutation score are the best metrics to quantify test suite effectiveness. Meanwhile, by using mutants instead of real faults, test effectiveness will be overestimated by more than 20% in values. **Conclusion.** We recommend that the standardized evaluation framework ASSENT should be used for evaluating and comparing test effectiveness metrics in the future work.

CCS Concepts: • **Software and its engineering → Software testing and debugging;**

Additional Key Words and Phrases: Test suite effectiveness, coverage testing, mutation testing, order preservation, statistical indicators

ACM Reference format:
Peng Zhang, Yang Wang, Xutong Liu, Zeyu Lu, Yibiao Yang, Yanhui Li, Lin Chen, Ziyuan Wang, Chang-ai Sun, Xiao Yu, and Yuming Zhou. 2024. Assessing Effectiveness of Test Suites: What Do We Know and What Should We Do?. *ACM Trans. Softw. Eng. Methodol.* 33, 4, Article 86 (April 2024), 32 pages.
https://doi.org/10.1145/3635713

1 INTRODUCTION

For the fault detection of software programs, assessing the effectiveness of test suites is one of the most fundamental tasks [3, 7, 8, 29]. Knowing the effectiveness of the current test suite can help the tester to develop the next test plan. To this end, many technologies have been proposed to evaluate the effectiveness of test suites. Coverage testing and mutation testing are the two most widely used technologies. For the coverage testing, a set of predefined covering requirements are first monitored when executing the **software under test** (**SUT**) against a test suite. Then, a code coverage is computed for the suite to measure its effectiveness. For mutation testing [21], a set of predefined mutators is first used to simulate the possible faults for the SUT, which are referred as mutants. Then, the test suite is executed to determine which mutants are killed (i.e., detected). After that, the effectiveness of the test suite is measured by computing the mutation score, which is the ratio of killed mutants to the total number of mutants.

Currently, researchers and practitioners are faced with a large number of metrics for evaluating test effectiveness. According to existing studies, mutation score is regarded as a more accurate metric than code coverage to estimate the effectiveness of a test suite [26, 37]. However, mutation testing is much more expensive than coverage testing as it requires more times of execution than the coverage testing to check each of those mutants. As a result, there are two main practices for improving the two technologies. For coverage testing, many fine-grained coverage criteria have been proposed for more accurate effectiveness evaluation [5, 6, 33]. For mutation testing, how to reduce the execution overhead, even if it may lead to the loss of accuracy, has become a research hotspot [13–16, 18, 20, 23, 25, 28, 30–32, 34, 36, 38, 42, 45, 47]. Regardless of their specific purpose, a practice on coverage testing provides a code coverage metric (e.g., branch coverage) while a practice on mutation testing provides a mutation score metric.

The central challenge lies in evaluating the accuracy of a test suite effectiveness metric. The most common practice is to perform correlation analysis on the independent variables (X, i.e., the metrics to evaluate) and dependent variables (Y, i.e., the ground truth). However, there are two main pitfalls: first, the description of the ground truth is too simplified, resulting in inaccurate assessment; second, data involving real defects is not suitable for analysis using traditional statistical indicators.

Let us take [39] as an example. Papadakis et al. took the following three steps to investigate the correlation between mutation score and real fault detection: (1) generate a bunch of test suites; (2) calculate metric values, such as mutation score, on them as X; and (3) execute them to obtain Y

(whether defects can be detected) as the ground truth. As a result, the authors found that the correlation between mutation score and fault detection was weak. However, **the meaning of "fault detection" as the ground truth is not clear enough.** In the context of "fault detection", our intuition is that a high fault detection represents a high effectiveness. Yet, after carefully scrutinizing the ground truth used in [39], we find that the authors use "fault detection" to mean that *given a number of test suites with the same size and a specific fault, the test suites that can detect this fault are more effective than the test suites that cannot detect this fault.* This raises a concern on the reliability of the ground truth deserves attention. Specifically, based on their ground truth, we can easily deduce that the triggering test case[1] is more effective than any other test case. For example, in Defects4J, for project "Chart", all the versions ID=3 (the latest version), ID=9, and ID=17 have bugs in the same class. If we assume that the three versions of the triggering test cases are t3, t9, and t17, we can deduce that t3 is more effective than t9 and t9 is as effective as t17 on version ID=3. However, such a deduction may require further analysis, since each of them actually prevents a specific real fault. If we change the version to ID=9, the deduction may not hold. As can be seen, **an oversimplified expression of ground truth can mislead readers into making intuitive deductions that may not align with what the author intended to convey.**

Moreover, **in the literature, it is common to see that inappropriate statistical indicators are used to measure the correlation between test effectiveness metrics and fault detection.** For instance, the commonly used Kendall rank correlation coefficient is used to measure the ordinal association between two measured quantities (X vs. Y). However, in Defects4J, there is often only one real defect for a version in a real defect dataset, making the dependent variable value (Y) for a test suite instance only have two values (0 or 1). In comparison, X, such as the coverage value, can be any rational number between 0 and 100. As a result, a larger number of tied values (i.e., a pair of same values) occur in Y. In [17], Gligoric et al. reported that 6.64% pairs of test suites were tied created using Branch Coverage-varied[2]. For real faults (Y), at least 50% pairs of test suites are tied. Ultimately, this leads to a misinterpretation of the correlation evaluation, with **a large number of tied values lowering the upper limit of the correlation coefficient, making it difficult for X to obtain a high correlation coefficient. Therefore, statistical indicators should be used with care when dealing with real defects.** Please refer to Section 3.3 for more details.

In summary, there is a pressing need to restructure the assessment system on test effectiveness metrics in our community. We can start with a high-level overview of an evaluation. First and foremost, researchers need to determine a ground truth to draw a fair conclusion for comparing the effectiveness among different test suites. The ground truth is used to answer the following question: when can we say that a test suite A is more effective than a test suite B in defect detection? For example, if A is the test suite that contains triggering test cases while B is the subset of A by removing those triggering test cases from A. We can say that A is more effective than B. This is because B is a subset of A, and A can detect a real defect while B cannot. Second, after determining the ground truth, researchers need to generate comparable A and B. Then, use the

[1]We follow the definition of [27]: for a specific fault under a certain version, we call this version the "buggy version". Meanwhile, there is at least one test case to reveal this fault. These failed test cases are called "triggering test cases". Then when the fault is fixed, we call the new version the "fixed version". Obviously, the triggering test cases will be passed on the fixed version.

[2]In [17], they used 2 two approaches to generate test suites: Coverage-varied and Size-varied. By coverage-varied, they created suites by uniformly selecting a coverage level between 1% and 100% and then randomly selecting tests from the test pool until they reached the selected level of a metric coverage. By this approach, the tied ratio ranges from 1.54% to 23.01% according to different coverage metric. For branch coverage, the tied ratio was 6.64%. By size-varied, they created 100 random suites for each size between 1 and 50. By this approach, the tied ratio ranges from 1.22% to 30.59% according to different coverage metric. For branch coverage, the tied ratio was 27.56%.

metric to derive the order in effectiveness for the same test suites. Lastly, the agreement between the two evaluations is calculated. For example, in [17], when comparing several coverage metrics using mutation score as the ground truth, the correlation coefficients between coverage and mutation score are calculated. In their study, the higher the coefficient, the more accurate the coverage metric is considered.

To summarize, the evaluation process needs to address the following three questions:

— When can we say that test suite A is more effective than test suite B? (ground truth)
— How to generate A and B? (benchmark test suites)
— Which indicator should be chosen to report the agreement between the **Metric To Evaluate (MTE**) and the ground truth? (agreement indicator)

We propose a framework ASSENT which stands for ev**A**luating te**S**t **S**uite **E**ffective**N**ess me**T**rics. **How do we use ASSENT?** In this article, we use ASSENT to evaluate the accuracy of seven test suite effectiveness metrics. To determine the ground truth order among test suites, we use the ability of different test suites in detecting real faults as the benchmark. Specifically, we use a test suite A with triggering test cases for a specific fault to filter out all the triggering test cases and obtain a subset B. We then say that A is more effective than B in terms of detecting that particular fault. This relation holds true not only for the buggy version but also for the fixed version of the program. The reason is twofold: for the buggy version, A can reveal the fault while B cannot; for the fixed version, the correctness is verified by the triggering test cases. To evaluate the metrics, we use the developer-written tests with triggering test cases as the test suite A for a buggy program. We then filter out the triggering test cases to obtain subset B. We compare the effectiveness of A and B using an MTE (e.g., coverage). If A achieves a higher MTE score than B, the ground truth order is preserved. However, if A achieves the same score as B, the ground truth order is twisted. We take all the faults into account, and the order-preserving score is then calculated to estimate how accurate a metric is in evaluating the effectiveness of test suites. Overall, ASSENT provides a reliable and effective way to evaluate the effectiveness of test suites using various metrics.

The contributions of our work include:

— We propose a practical framework ASSENT for evaluating the accuracy of test effectiveness metrics. With ASSENT, we are able to identify which test suite effectiveness metric is more accurate.
— We analyze the possible threats of those ground truths in existing studies and propose a new ground truth based on real defects.
— Based on ASSENT and the proposed ground truth, we compare the accuracy of a variety of representative test suite effectiveness metrics. Our data and code are publicly available [1].

The rest of this article is organized as follows. Section 2 introduces the background of our study. Section 3 presents the framework ASSENT we proposed. Section 4 describes the settings of experiments. Section 5 reports the experimental results. Section 6 summarizes the recommendations and the implications. Section 7 analyzes the threats to the validity. Section 8 concludes the article.

2 BACKGROUND

In this section, we provide an overview of the most commonly used testing practices for generating mutation score metrics and code coverage metrics, as well as related work in metric evaluation.

2.1 Mutation Testing

In previous studies, the following mutation reduction strategies are commonly used to reduce the number of mutants before a mutation score metric is computed.

— ***Certain Operator Selection (COS)*** [13–15, 34, 36, 42]: select the mutants generated by a specific mutator subset to compute the mutation score. There are numerous practices on COS, the core idea of which is to remove redundant mutation operators and select important operators that are more related to defects. Mathur first proposed COS in 1991 and called it selective mutation [34]. They found that the number of redundant mutants could be reduced by 30% to 40% by deleting "arithmetic operator replacement" mutator and "scalar variable replacement" mutator. Offutt et al. found five important mutation operators for COS, including the relational, logical, arithmetic, absolute, and unary insertion operators [36].
— ***Random Mutant Selection (RMS)*** [31, 38, 45]: randomly select a specified number or proportion of mutants from all mutants to compute the mutation score metric.

Beside mutant reduction, other practices aim at providing more accurate metrics. These metrics are performed after all mutants are executed.

— ***Subsuming Mutation Score (SMS)*** [16, 18, 30, 32]: select the subsuming mutants to compute the mutation score. SMS is proposed for avoiding inflation. By the definition in [30], "One mutant subsumes another if at least one test kills the first and every test that kills the first also kills the second". For all mutants, killing the subsuming mutants is to kill all the killable mutants. Therefore, the subsumed mutants should be regarded as redundant. In [30], Kurtz et al. proposed both dynamic and static methods to approximate the "true" subsumption. In [18], Gong et al. manually inserted mutant branches into the original program to generate a giant program which was used to analyze the approximation of the "true" subsumption.
— ***Clustering Mutation Score (CMS)*** [23]: select one mutant from each mutant cluster to compute the mutation score. First, a number of features are collected among all the mutants. Second, based on these features, a clustering algorithm is used to group the mutants into different clusters. Third, the selection is generated by selecting one mutant from each mutant cluster. For example, Hussain [23] executed all the mutants to obtain the information on whether a test can kill a mutant against all the tests and mutants. After that, K-means and agglomerative clustering were applied to cluster the mutants. By the clustering algorithm, the mutants in the same cluster were guaranteed to be killed by a similar set of test cases. Therefore, randomly selecting one mutant from each cluster can be seen as a selection of representative mutants.

2.2 Code Coverage Testing

There are several coverage metrics that can be used to measure the level of code coverage, including statement coverage, branch coverage, and so on.

— ***Statement coverage (SC.)*** SC is a code coverage metric that measures the percentage of executable statements in a program that have been executed during testing. It indicates the proportion of code that has been exercised by the test suite. In other words, it measures the completeness of a test suite in terms of statements covered. A statement is considered covered if it is executed at least once during testing.
— ***Branch coverage (BC.)*** BC is another code coverage metric that measures the percentage of program branches that have been executed during testing. It determines the proportion of decision points in a program that have been executed during testing. A branch is considered covered if both the true and false branches have been executed at least once during testing.

Of course, there are also more fine-grained coverage metrics such as DBB coverage [6] and Intra-method coverage [11]. Their monitoring path is more complex. Meanwhile, they are not

compared in this article. The main reason is that the extensive comparisons of coverage metrics have already been conducted in [17], and this article focuses more on mutation metrics. Moreover, their experimental results show that BC is already comparable to other computationally expensive coverage metrics. Therefore, using BC as a representative of coverage metrics is reasonable.

2.3 Test Effectiveness Metric Evaluation

In Table 1, we summarize and compare representative studies related to the core elements of our framework ASSENT: "ground truth", "benchmark test suites", and "agreement indicator". As can be seen, with such a framework, we can easily compare the assumptions and processes of the relevant work. This is one of our major motivations for proposing a standardized evaluation framework. From the table, we can see that the primary reason for the controversy in existing research is that different ground truths are used. As a result, we recommend that the question on ground truth should be answered explicitly in future work. From their findings, we can find that some of the conclusions are contradictory. For example, from [39], we know there is a weak correlation between mutation score and real fault detection, while from [27] there is a strong correlation between mutation score and real fault detection. In this article, we attribute these different findings to the variations in the "ground truth", "benchmark test suites", and "agreement indicator" used in each study.

It should be noted that a few studies were not focused on correlation analysis. For example, early research in the field often began with a theoretical approach to examine the relationship between MTEs [44, 46]. Various relationships such as "BETTER (in practice it is often the same as subsumption)", "properly covers", "universally covers" and others were defined to assess the effectiveness of testing criteria [44]. However, it became evident that not all MTEs hold such relationships with each other. As a result, it is difficult to categorize most MTEs as either strong or weak based solely on this type of study. For another example, [10] did not investigate the agreement between the MTE and the ground truth. Instead, they examined which MTE is the most effective way to select test cases. Another example is when researchers investigate a new mutation testing technique (i.e., a mutation score MTE against a mutant set), they often use the following evaluation approach [4, 43]: first, find the minimal set of test cases that kills the set of mutants, and then evaluate the quality of this MTE based on the effectiveness of this minimal test suite. The higher the effectiveness of this minimal test suite, the higher the quality of this MTE. This approach is commonly used. However, there are two issues. First, there is randomness in the process of finding the minimal set, for instance, in the case of a single mutant, there may be several test cases that kill it, most of which are likely to simply cover it eventually leading to a kill; Second, it is difficult to generalize this evaluation. For example, if the comparison object is extended to coverage, how should it be compared at the same time? Given that these evaluations are not general in nature, in this article, we will focus on the agreement calculation as the mainstream study.

This article draws inspiration from [17], which provides a guideline for comparing coverage using a statistical approach. Building on this, our aim is to develop a guideline for comparing multiple MTEs. We reviewed several statistical approach articles [17, 19, 22, 24, 29, 35, 39, 41], but found that their conclusions varied significantly. Therefore, we asked the question, what are the reasons behind these discrepancies?

The above question is the primary driver of our research and distinguishes our article from existing studies. Our first goal is to identify and clarify the critical factors that may impact the results of previous studies and develop an evaluation framework that centers around these core factors. In the following sections, we summarize the key insights gained from prior studies and present an evaluation framework that researchers can use to develop a thorough and efficient evaluation plan for comparing any test effectiveness metrics.

Table 1. Related Work

Article	Year	Venue	Ground Truth	Benchmark Test Suites	Agreement Indicator	Summary of Scientific Findings
Namin et al.	2009	ISSTA	|A| = |B|, MS(A) > MB(B)	T is the test pool. A B are randomly sampled of fixed size from T	Kendall Pearson R^2	For small suites, there is a moderate to very high correlation between mutant detection ratios and coverage. For large test suites, there is a low to moderate correlation
Just et al.	2014	FSE	A ≈ B, FD(A) > FD(B)	B passes on buggy and fixed version. A fails on buggy version but passes on fixed version. A and B differ by one modified or added test.	OP	There is a strong correlation between mutant detection ratio and real fault detection ratio
Inozemtseva and Holmes	2014	ICSE	|A| = |B|, MS(A) > MS(B)	T is the test pool. A B are randomly sampled of fixed size from T	Kendall Pearson	There is a negligible to moderate correlation between mutant detection ratio and statement coverage
Gopinath et al.	2014	ICSE	MS(A)> MB(B)	Each test set(developer-written and automatically -generated) was used for analysis without sampling.	Kendall R^2	There is a strong correlation between mutant detection ratio and statement coverage.
Kochhar et al.	2015	SANER	|A| = |B|, FD(A) > FD(B)	T is the test pool. A B are randomly sampled of fixed size from T	Point Biserial Correlation	There is a moderate to strong correlation between mutant detection ratio and real fault detection ratio.
Gligoric et al.	2015	TOSEM	|A| = |B|, MS(A) > MB(B)	A B are generated by keeping randomly selecting test cases until the MTE value is in a given interval.	Kendall Pearson R^2	There is a strong correlation between mutant detection ratio and branch coverage.
Chekam et al.	2017	ICSE	B|, FD(A) > FD(B)	T is the test pool. A B are randomly sampled of fixed size from T	–	There is a strong connection between mutation score increase and fault detection at higher score levels
Shin et al.	2018	TSE	FD(A) > FD(B)	A B are generated by keeping greedily selecting test cases until the MTE value is in a given interval.	Rank-biserial Correlation	Distinguishing mutation adequacy criterion has higher fault detection probability than strong mutation adequacy criterion.
Papadakis et al.	2018	ICSE	|A| = |B|, FD(A) > FD(B)	T is the test pool. A B are randomly sampled of fixed size from T	Kendall Pearson	There is a weak correlation between mutant detection ratio and real fault detection ratio.
Hariri et al.	2019	ICST	|A| = |B|, FD(A) > FD(B)	T is the test pool. A B are randomly sampled of fixed size from T	Kendall Pearson	There is a very weak correlation between mutant detection ratio and real fault detection ratio
Chen et al.	2020	ASE	FD(A) > FD(B)	Greedily adding one test into B to achieve a higher adequacy for a MTE to obtain A	Probabilistic Coupling	Adequacy-based test selection is superior to random selection. Mutation-based test selection is most effective when employed after coverage has exhausted its usefulness.
Our study	2024	TOSEM	B⊊ A, FD(A) > FD(B)	A fails on buggy version but passes on fixed version. B is obtained by deleting triggering test cases from A	OP	Mutation score and subsuming mutation score are the best metrics

The "ground truth" is when they assume A is more effective than B. "FD" is the number of faults detected by the corresponding test suite. "MS" is the mutation score of the corresponding test suite. "OP" is Order Preservation (explained in Section 3.3).

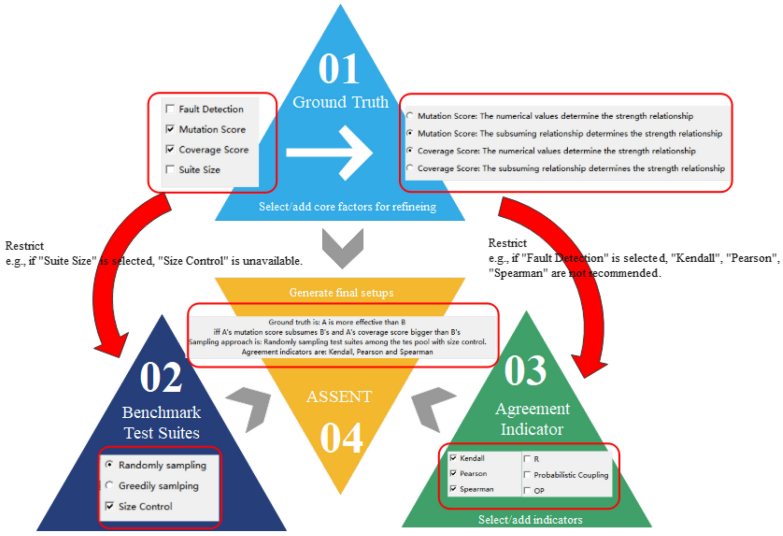


Fig. 1. ASSENT for the evaluation of test effectiveness metrics. The three triangles depicted in the figure represent the three elements of ASSENT, which correspond to the three steps involved in designing experiments using the framework. In particular, the red box highlights a screenshot of a case where ASSENT was utilized to design an experiment. For a more detailed explanation of this case, please refer to Section 3 of the article. The **graphical user interface** (**GUI**) of ASSENT is available in the repository and can be used for experimental design purposes.

3 ASSENT: EVALUATING TEST SUITE EFFECTIVENESS METRICS

The purpose of ASSENT is to evaluate the extent to which a given test effectiveness metric (note that it is preferable for this metric to differ from the ground truth.) can replace the ground truth, i.e., agreement. The higher the agreement indicator, the closer the test suite evaluation under the test effectiveness metric is to the evaluation under the ground truth.

ASSENT framework consists of three core elements. Firstly, it requires an explicitly defined ground truth, which serves as the basis for evaluating the effectiveness of test suites. Secondly, benchmark test suites are utilized to calculate empirical relations with respect to the ground truth, as well as numerical relations under the chosen metric for evaluation (referred to as MTE). Thirdly, indicators are employed to determine the agreement between the empirical and numerical relations.

Figure 1 illustrates how ASSENT framework evaluates the accuracy of a test suite effectiveness metric using an example. In this example, we initially select "Mutation Score" and "Coverage Score" as the factors for the ground truth. Subsequently, we utilize the numerical value in the coverage score and the concept of subsuming relationship to further refine the ground truth. Moving on to the second step, we choose "Randomly Sampling" and "Size Control" as the options. Finally, in the third step, we specify the statistical indicators to be used. ASSENT then generates the experimental setups accordingly. For instance, an experimental setup could include the following specifications: 'Ground truth is: A is considered more effective than B iff A's mutation score subsumes B's and A's coverage score is greater than B's. Sampling approach is: Randomly sample test suites from the test pool with size control. Agreement indicators are: Kendall, Pearson, and Spearman'.

It is worth noting that ASSENT not only provides predefined options but also allows researchers to add their own ground truth factors and agreement indicators in the first and third steps,

respectively. Furthermore, it is important to be aware that certain limitations and recommendations exist among the options within ASSENT. For example, if "Suite Size" is selected as the ground truth in the first step, the option to control the size will no longer be available in the second step. Another example is that if multiple factors are chosen in the first step, only the "**Order Preservation (OP)**" indicator can be used in the third step.

Overall, ASSENT offers flexibility and customization while considering the limitations and recommendations within its options.

3.1 Ground Truth Determination

Let us imagine a specific question as the motivation example for deriving the empirical relationships between test suites: given two test suites A and B of an SUT , when can we say that A is more effective than B in terms of defect detection? Here are some possible answers:

Proposition 3.1. *If B is a proper subset of A, A is deemed as more effective than B.*

Proposition 3.2. *If A achieves a higher code coverage than B, A is deemed as more effective than B.*

Proposition 3.3. *If A achieves a higher mutation score than B, A is deemed as more effective than B.*

Proposition 3.4. *For a given real fault set, if the fault set detected by B is a proper subset of that by A, A is deemed as more effective than B.*

.....

The purpose of this example is to make the ground truth criterion concrete rather than simply discussing these propositions. These propositions summarize the empirical knowledge of the relationship among test suites. By taking some propositions, we can design the ground truth criteria to derive the ground truth order among test suites. It should be noted that this article does not adopt only one proposition.

After taking a brief introduction of the ground truth criterion, the following section includes two parts: first, we discuss the possible threat of the ground truth used in the existing studies; second, we propose the ground truth used in this article.

3.1.1 Threat of Existing Studies. In this subsection, we will analyze the possible threat of representative studies that comes from the ground truth. At the high level, the threat of using simulated faults (i.e., mutants) as the ground truth is obvious, which mainly comes from the difference between the mutants and real defects. However, there is still one question that has not been answered: will this approach lead to overestimation or underestimation of the MTE? Therefore, our experiment answers this question. In this subsection, we focus on the threat of using real faults as the ground truth.

In [39], researchers found that there was a weak correlation between mutation score and real fault detection when controlling the test suite size. For the test suites of the same size, they assumed that those which can detect the real fault were more effective than the others. This ground truth is implied by the indicator of the Kendall coefficient. It seems that they not only controlled the size but also used real defects. As a result, their research looks "solid". However, we can use a simple example to illustrate the pitfall of its "size". Suppose that for a particular fault, there is only one triggering test case t in the test pool T. We let A = {t} and B = {t'} where t' $\in$ T/{t}. According to their ground truth, A is more effective than B. Therefore, once B obtains a higher mutation score than A, a penalty will be counted when calculating the Kendall coefficient. So if we go back to the assumption, we can find that although t can detect the specific fault, we cannot assume that

another test case is less effective. It is possible that another test case was used to detect another fault (e.g., a fixed fault in the previous version). For example, in Defects4J, for project "Chart", the versions ID = 3, ID = 9, and ID = 17 all have bugs in the same class with different triggering test cases. Therefore, the assumption that A is more effective than B will be questioned when the possibility of multiple defects is considered (e.g., the intention of mutation testing).

In [27], researchers found a strong correlation between mutant detection ratios and real fault detection ratios. They assumed that A was the test suite that failed on the buggy version but passed on the fixed version, while B was the test suite that passed on both the buggy and fixed versions. Meanwhile, A and B differ by one modified or added test. They drew the conclusion that A was more effective than B. Their logic was that A could detect specific defects, while B could not. However, if the difference between A and B comes from the "modified test", the same threat as in [39] will happen: the modification will be effective for the specific defect. Meanwhile, it cannot be denied that it is more effective for another possible defect before modification. If the difference comes from the "added test", the ground truth will be solid. As a result, we propose this ground truth in this article.

3.1.2 Proposed Ground Truth. Combining Propositions 1 and 4, when comparing two test suites A and B, the following proposition (i.e., assumptions) is held:

If $B \subseteq A$, then

$$A \text{ is more effective than } B \Leftrightarrow FD(B) < FD(A). \tag{1}$$

Here, FD(T) is the number of faults detected by T (i.e., A or B here). Because B is the subset of A, we can conclude that A cannot be less effective than B. Then, Proposition 4 states that if, based on a specific dataset, we observe that A detects a greater number of faults compared to B, we believe that A has a stronger effectiveness in terms of defect detection. In other words, even for faults that have not yet been observed, we believe that A has a better capability to detect them.

This is the ground truth adopted in this article. We chose "fault detection" and "size" as the ground truth for two reasons. Firstly, "fault detection" is the most commonly used and widely accepted criterion in the literature. Secondly, introducing the constraint of "size" (i.e., subset) ensures that the relative strengths among all the compared test suites hold 100% confidence. This addresses potential issues such as conclusions could shift between different versions of "Chart" by considering only "fault detection".

Given that real faults may be expensive or hard to obtain in practice, we also compare the impact of using mutants instead of the real faults in our experiment:

Alternative ground truth Combing Propositions 1 and 3, the following proposition is held:

If $B \subseteq A$, then

$$A \text{ is more effective than } B \Leftrightarrow MS(M, B) < MS(M, A). \tag{2}$$

Here, M is the whole possible mutant set, while MS(M, T) is the mutation score computed when executing T against M. Assume that KM(M, T) is the set of mutants in M killed by T and |M| is the number of mutants in M. Then, MS(M, T) = |KM(M, T)|/|M|. The reason for choosing the mutation score over the coverage score is that most past studies support the assumption or result that the mutation score is better [9, 10, 17, 27]. Note that the alternative ground truth is the baseline that is used for analyzing the impact of mutants.

After analyzing the threat of the existing ground truth and proposing our ground truth, we will introduce the approach to generating test suites for comparison in the next subsection.

3.2 Benchmark Test Suite Generation

We can summarize the process of generating benchmark test suites in two steps: the first step involves creating a pool of test cases, while the second step involves sampling test cases from that pool.

3.2.1 Generating the Test Pool. In the first step, researchers use manually written test cases and/or automatically generated test cases to obtain the entire pool. However, there are significant differences between these two types of test case collections, and this may be an important reason for the inconsistent conclusions of many studies.

In 2019, Serra et al. [40] compared the performance of manually and automatically created test suites using three automatic test generation tools: EvoSuite, Randoop, and JTExpert. They considered three aspects: code coverage, mutation score, and fault detection capabilities. Their results showed that: (1) current automatic test case generation tools can outperform manually written tests in obtaining higher coverage and mutation scores; and (2) manually written tests can have higher defect detection capability than automatic tools.

Furthermore, the situation may be even more concerning as automatic test generation tools may also produce harmful test cases. For instance, the popular tool EvoSuite has a crucial feature that assumes the correctness of the source program and generates assertions accordingly. However, if the current version of the code contains bugs, EvoSuite will generate assertions based on the erroneous behavior, which can result in harmful test cases. To illustrate, when generating test cases on the buggy version of Defects4J using EvoSuite (by running "gen_tests.pl -g evosuite -p Lang -v 1b"), the resulting test suite may contain test cases that pass on the faulty version and fail on the fixed version. We consider such test cases to have a negative impact on the effectiveness of testing.

Therefore, when using real defects as ground truth, it may not be appropriate to use automated tools to build test case pools, as they may include harmful test cases and weaken the ability to detect defects. In this article, besides RQ4, we consider the real defect as the ground truth and do not include automatically created test cases in the pool. In RQ4, we use automatically generated test cases to investigate the difference between RQ1.

3.2.2 Sampling from the Pool. The second step is sampling from the pool. The approach to sampling depends on the ground truth. In many studies, researchers only compare test suites of the same size by their ground truth in order to eliminate the confounding effect of size. However, the question is whether we should control the test suite size.

In [10], Chen et al. explained why test suite size is neither a confounding variable nor an independent variable that should be experimentally manipulated. Their opinion is that no tester will use "size" as the testing target, and we agree with this opinion. Besides, we would like to add that it may be difficult to compare test suites of the same size. For the simplest example, for two different test cases, t1 and t2, it is hard to conclude that t1 is more effective than t2 in most cases. The main reason is that they are most likely designed for different test targets.

Therefore, we decided to use different sizes to provide a solid ground truth, as there is no basis for controlling the size to eliminate the confounding effect. Adding test cases to a test suite does not decrease its effectiveness[3], so it is safe to compare test suites of different sizes by real faults. This approach is also in line with the actual testing scenario, where testers incrementally expand test suites to achieve adequacy for a given MTE criterion.

[3]The assumption we rely on here is that testers will not add harmful test cases into the test suite. As we pointed out in Section 3.2, it is difficult for automation tools to avoid threat by harmful test cases. To this concern, by using manual test case sets, such problems are avoided as much as possible.

This approach is also in line with the actual testing scenario. In [10], to simulate the manual testing, Chen et al. "greedily select one test at a time to incrementally achieve adequacy for a given adequacy criterion" (i.e., MTE in our context). In essence, this is a process of continuous expansion of test suites, which is consistent with our approach.

3.2.3 Generation in This Article. Now we will describe how to generate test suites in this article. By the well-known benchmark Defects4J, for each fault, there is a buggy version and a fixed version, and a manually written test suite A that passes the fixed version, which includes at least one triggering test case that fails only for the buggy version. We consider A as the initial test suite for the specific defect. We then obtain a test suite B by filtering out all the triggering test cases from A, i.e., B is the set of non-triggering test cases in A. We assume that A is more effective than B in terms of detecting the specific defects not only for the buggy version but also for the fixed version. The reason is twofold: for the buggy version, A can reveal the defects while B cannot, and for the fixed version, the correctness is verified by the triggering test cases.

Moving on, it is important to consider whether we should take into account additional relationships between test suites, beyond A and B. For instance, we may wonder whether t1, t2 is more effective than t1, where t1 is a non-triggering test case and t2 is a triggering test case. It is true that t1, t2 can reveal a defect that t1 cannot, but this relationship is easier to discern by most metrics because adding more test cases will generally increase value. For example, in most cases, two test cases will achieve a higher BC than any one of the two test cases. In essence, it is the increase in test suite size that leads to an increase in coverage. For most criterion, it is easy to distinguish t1, t2 from t1 because of the influence of size. To this concern, we need to eliminate the confounding effect as much as possible. Meanwhile, if a criterion can distinguish between A and B, for any subset of B, namely, C, this criterion must be able to distinguish between C∪(A-B) and C. This is because A-B can uniquely contribute to the increase of the criterion metric. Therefore, it is not necessary to take C into consideration.

To achieve this purpose, we compare the complete set A with the subset B, which ensures that the relative size is as similar as possible. For example, if A has 100 test cases and only one triggering test case, the difference in size between A and B will be only 1%. This way, if a criterion can accurately reflect the relationship between A and B, it should also be sensitive enough to distinguish between subsets of B and A-B. On the other hand, if a criterion cannot differentiate between A and B (e.g., they achieve the same BC), then the set of triggering test cases is useless for that criterion. From another perspective, it can also be considered that we have posed a challenge to all metrics: can they still identify the increase in effectiveness under the condition that the tests already have a certain level of strength?

Therefore, we focus only on comparing the effectiveness between A and B as the ground truth, which ensures that the relative size is as similar as possible. This allows us to better understand the metrics' measurement ability.

3.3 Agreement Calculation

Once we have generated the test suites for comparison, we need to select an appropriate measure to assess the agreement between the MTE and the ground truth. Ideally, the best metric should be consistent with the ground truth. Typically, we use an indicator to report this agreement.

Correlation analysis is a common approach to evaluate MTE. Researchers calculate the scores provided by the MTE for each pair of (A, B) and then compute the correlation coefficient, such as Spearman correlation, Pearson correlation, or Kendall correlation, between the MTE order and the ground truth order. When interpreting the calculated correlation values, researchers often rely on conventions from other fields. Cohen has the following interpretation of the absolute value of the correlation [12]:

Table 2. Interpretation of the Absolute Value of the Correlation

Correlation coefficient value	Association
−0.3 to 0.3	Weak
−0.5 to −0.3 or 0.3 to 0.5	Moderate
−0.9 to −0.5 or 0.5 to 0.9	Strong
−1.0 to −0.9 or 0.9 to 1.0	Very strong

For example, in [39], when the correlation values were mostly clustered between 0.35 and 0.75, they were interpreted as moderate to strong correlation. When the correlation values were clustered between 0.05 and 0.20, they were interpreted as weak correlation. This interpretation method was borrowed from other fields and is consistent with people's intuitive cognition. However, **in our specific scenario, the proportion of tied pairs in Y is much higher than that in X, which will result in a lower upper bound for the correlation between X and Y. In a word, the theoretical upper bound of 1.0 for the correlation coefficient is almost impossible to achieve.** Therefore, more adjustments are needed in the interpretation. In the following, we will explain these pitfalls in detail.

3.3.1 Kendall Rank Correlation Coefficient. Kendall rank correlation coefficient Tau is used to measure the ordinal association between X and Y. The range of Tau is [-1,1]. 0 means the two variables are completely unrelated. 1 means the two variables are completely positively correlated. There are three formulas used to calculate Tau, which are called Tau-A, Tau-B, and Tau-C:

$$\tau_A = \frac{n_c - n_d}{n_0}, \tag{3}$$

$$\tau_B = \frac{n_c - n_d}{\sqrt{(n_0 - n_1)(n_0 - n_2)}}, \tag{4}$$

$$\tau_C = \frac{2(n_c - n_d)}{n^2 \frac{m-1}{m}}. \tag{5}$$

Where

$$n_0 = \frac{n(n-1)}{2}, \tag{6}$$

$$n_1 = \sum_i \frac{t_i(t_i - 1)}{2}, \tag{7}$$

$$n_2 = \sum_j \frac{u_j(u_j - 1)}{2}, \tag{8}$$

n = Number of data in X
n_c = Number of concordant pairs
n_d = Number of discordant pairs
t_i = Number of tied values in the ith group of ties for the first quantity
u_j = Number of tied values in the jth group of ties for the second quantity
r = Number of rows
c = Number of columns
$m = \min(r, c)$

We will explain each definition in detail with an example. Before that, let us first show that Tau-A and Tau-C are almost equivalent in our scenario. Here, r and c represent the range of values of

Table 3. Different Upper Bound of Tau Under Different Tied Ratio in X

Total pairs	Tied in Y	Tied Ratio in X	Tied in X	n_c	n_d	Tau-a (upper bound)	Tau-b (upper bound)
499500	249500	0%	0	249000	0	0.499	0.707
499500	249500	5%	24950	249000	0	0.499	0.725
499500	249500	20%	99800	249000	0	0.499	0.790
499500	249500	50%	249500	249000	0	0.499	1.000
499500	396000	5%	24950	99000	0	0.200	0.459

X and Y, respectively. In the defect detection scenario, there are only two values for Y. As a result, the values of c and m are both 2. This means that the denominator of Tau-C is n^2. When n tends to infinity, Tau-A and Tau-C are converge to each other. Therefore, we will focus on analyzing Tau-A and Tau-B in the following explanations.

Next, let us illustrate the pitfall of using Tau through an example. At a high level, we aim to demonstrate **how the upper bound of Tau between X and Y varies with different proportions of tied pairs of X, given that the proportion of tied pairs of Y is minimal**[4]**.** Firstly, we need to determine the minimum tied pair ratio of Y. For instance, let us consider a scenario where there are 1,000 test suites, with d test suites detecting the faults and 1,000-d failing to detect the faults. Thus, the number of untied pairs is d*(1,000-d)<=500*500. The ratio of untied pairs is less than 500*500/(1,000*999/2) = 50%. As a result, the minimal ratio of tied pairs in Y is 50%.

Then we calculate the value of Tau by scaling the proportion of tied pairs in X from 0%, 5%, 20%, and 50%. At the same time, we assume the number of discordant pairs is zero to obtain the maximal Tau. The results are presented in the table above. We observe that **even if 20% of tied pairs appear in X, the upper bound of Tau-b is 0.790 instead of 1.0. If a theoretical upper bound of 1.0 is desired, then X must have the same number of tied pairs as Y.** However, considering that only 6.64% of pairs of test suites are tied when using BC-varied, as reported in [17], we need to redefine the usage of Tau-b on real faults.[5]

3.3.2 Spearman's Rank Correlation Coefficient. Spearman's rank correlation coefficient ρ assesses how well the relationship between X and Y can be described using a monotonic function. The range of ρ is [-1,1]. 0 means the two variables are completely unrelated. 1 means the two variables are completely positively correlated. The computation formula is as follows:

$$\rho = \frac{\text{cov}(\text{Rank}(X), \text{Rank}(Y))}{\sigma_{\text{Rank}(X)}\sigma_{\text{Rank}(Y)}}.$$

Where:

— $\text{Rank}(X)$ is the rank variable of $X : x_i (i = 1, 2, \ldots, n)$.
— $\text{Rank}(Y)$ is the rank variable of $Y : y_i (i = 1, 2, \ldots, n)$.
— $\text{cov}(\text{Rank}(X), \text{Rank}(Y))$ is the covariance of the rank variables.
— $\sigma_{\text{Rank}(X)}$ is the standard deviations of the rank variable.

Then:

$$\rho = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}.$$

[4]We minimize tied pairs in Y for the maximization of Tau. In the last row in Table 4, we can see that when the tied ratio of Y is 80%, the upper bound of Tau will decrease.
[5]Note that in [17], there is no such pitfalls, since they investigated the correlation between coverage and mutation scores: 6.31% of the test suites are tied by mutants, which is close to coverage.

For Spearman's correlation coefficient, the situation is a little more complicated. We prove the following theorem: if there are 50% tied pairs in Y and there is no tied pair in X, the upper bound of the correlation coefficient between X and Y is the $\frac{\sqrt{3}}{2}$ instead of 1.

Suppose there are $n = 2k$ test suites. First, since there is no tied pair in X, the rank values of X are $[1, 2, \ldots, 2k]$. Second, by computing average rank, we can obtain the rank values of Y are $[\frac{(k+1)}{2}]$ and $[\frac{(3k+1)}{2}]$ each for k test suites, respectively. We can get that the average value of rank of X or Y is $(2k+1)/2$. From the Rearrangement Inequality,[6] we have:

$$
\begin{aligned}
\rho &= \frac{\sum_{i=1}^{n}(x_i-\bar{x})(y_i-\bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i-\bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i-\bar{y})^2}} \\
&\leq \frac{\sum_{i=1}^{k}\left(i-\frac{2k+1}{2}\right)\left(\frac{k+1}{2}-\frac{2k+1}{2}\right)+\sum_{i=k+1}^{2k}\left(i-\frac{2k+1}{2}\right)\left(\frac{3k+1}{2}-\frac{2k+1}{2}\right)}{\sqrt{\sum_{i=1}^{n}(x_i-\bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i-\bar{y})^2}} \\
&= \frac{\sum_{i=1}^{k}\left(i-\frac{2k+1}{2}\right)\left(-\frac{k}{2}\right)+\sum_{i=k+1}^{2k}\left(i-\frac{2k+1}{2}\right)\left(\frac{k}{2}\right)}{\sqrt{\sum_{i=1}^{n}(x_i-\bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i-\bar{y})^2}} \\
&= \frac{\frac{k^3}{2}}{\sqrt{\sum_{i=1}^{n}(x_i-\bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i-\bar{y})^2}} \\
&= \frac{\frac{k^3}{2}}{\sqrt{\left(\sum_{i=1}^{n}(x_i-\bar{x})^2\right)\left(\sum_{i=1}^{2k}\left(\frac{k}{2}\right)^2\right)}} \\
&= \frac{\frac{k^3}{2}}{\sqrt{\frac{k^3}{2}\sum_{i=1}^{2k}(x_i-\bar{x})^2}} \\
&= \frac{\frac{k^3}{2}}{\sqrt{\frac{k^3}{2}\sum_{i=1}^{2k}\left(i-\frac{2k+1}{2}\right)^2}} \\
&= \frac{\frac{k^3}{2}}{\sqrt{\frac{k^3}{2}\sum_{i=1}^{2k}\left(i-k-\frac{1}{2}\right)^2}} \\
&= \frac{\frac{k^3}{2}}{\sqrt{\frac{k^3}{2}\left[\sum_{i=1}^{2k}(i-k)^2-\sum_{i=1}^{2k}(i-k)+\sum_{i=1}^{2k}\frac{1}{4}\right]}} \\
&= \frac{\frac{k^3}{2}}{\sqrt{\frac{k^3}{2}\left[\left(\sum_{i=1}^{2k}(i-k)^2\right)-k+\frac{k}{2}\right]}} \\
&= \frac{\frac{k^3}{2}}{\sqrt{\frac{k^3}{2}\left[2\left(\sum_{i=1}^{k}i^2\right)-k^2-\frac{k}{2}\right]}}
\end{aligned}
$$

[6]Rearrangement Inequality: for two sequences $a_1 \leq a_2 \leq \cdots \leq a_n$, $b_1 \leq b_2 \leq \cdots \leq b_n$, the inequalities $a_1b_n + \cdots + a_nb_1 \leq a_1b_1 + \cdots + a_nb_n \leq a_1b_{\pi(1)} + \cdots + a_nb_{\pi(n)}$ hold, where $\pi(1), \pi(2), \ldots, \pi(n)$ is any permutation of 1, 2, ..., n.

$$= \frac{\frac{k^3}{2}}{\sqrt{\frac{k^3}{2}\left[\frac{k(k+1)(2k+1)}{3} - k^2 - \frac{k}{2}\right]}}$$

$$= \frac{\frac{k^3}{2}}{\sqrt{\frac{k^4}{2}\left[\frac{(2k-1)(2k+1)}{6}\right]}}$$

$$\lim_{k\to\infty} \frac{\frac{k^3}{2}}{\sqrt{\frac{k^4}{2}\left[\frac{(2k-1)(2k+1)}{6}\right]}} = \lim_{k\to\infty} \frac{\frac{1}{2}}{\sqrt{\frac{1}{2} * \frac{4}{6}}} = \frac{\sqrt{3}}{2}$$

We can observe that Spearman's p also faces the same issue as Tau, where the upper bound of the correlation coefficient is not always 1. This means that if we rely on interpretations from other areas, the resulting "strong" or "weak" relationships may not have the expected explanatory power. We are still investigating the upper bound of the correlation coefficient for the more general case where there is a tied pair with a ratio of r_y in Y and a tied pair with a ratio of r_x in X.

What are the reasons for the pitfalls happening on the most commonly used rank coefficients? From the computation of Tau or ρ, we can find that a metric is also required to give the same score to the set of test suites that can detect defects, or else obtain a penalty. This assessment practice potentially lowers the upper limit of the correlation coefficient which is unreasonable.

With some simple examples like above, we can see how the misuse of statistical indicators can lead to a misleading conclusion. Of course, the example we constructed is highly contingent. For instance, there may be a large difference in the number of detected test suites and the number of undetected suites. And the MTE values may also have many tied values. However, the following fact is general:

1. Since the number of real faults is small, the range of values for Y is much smaller than the range of values for X, which will eventually lead to the occurrence of much more tied values in Y than tied values in X. For X, in [17], they reported that 6.64% pairs of test suites are tied created using BC-varied. For Y, we have proved that the ratio of tied pairs is more than 50%.
2. For the tied values in Y, if X cannot provide tied values, respectively, the metric will receive a penalty in the calculation of the correlation.
3. The tied values in Y indicate a group of test suites that can detect the same number of actual faults. It is important to note that tied values do not necessarily mean that these test suites are equally effective; rather, it simply means that it is impossible to differentiate between these test suites based on specific real defects. Therefore, they may not necessarily require the same score. Within these suites, it is possible that some test suites may cover more program behaviors and protect against more potential defects, even if those defects do not actually occur.

3.3.3 Modified Indicator. From the above example, we can see that it is reasonable to consider the strength relationship between non-tied test cases. For tie values, it is best to filter them to avoid situations where ground truth cannot determine which test suite is more effective. In this article, we use a variant of the Kendall coefficient as the indicator to calculate the agreement. Specifically, in [48], Zhang et al. proposed an indicator OP to quantify the "order-preserving ability" of mutant reduction strategies. Given a mutation reduction strategy, OP measures the extent to which the mutation score orders among test suites are maintained before and after mutation reduction. In this article, we adapt OP to quantify the ability of a test effectiveness metric in maintaining the ground truth order among test suites.

According to a ground truth criterion, we suppose that we have generated p test suite pairs with the order by faults. In our context, we assume that there is only one defect in a buggy version. This assumption is consistent with the existing benchmark (i.e., Defects4j). We use (x, y, operator) to represent a pair of ordered test suites. x and y are the pair of test suites. The operator is "more effective than" if x is more effective than y under a specific metric. We check whether each (x, y, operator) is satisfied or not under the test suite effectiveness metric m:

$$\text{check}(x, y, \text{operator}, m) = \begin{cases} 1 & \text{if } (x, y, \text{operator}) \text{ is held by } m \\ 0 & \text{else.} \end{cases}$$

Furthermore, we define:

$$W = \{(x, y, \text{operator}) \mid \text{check}(x, y, \text{operator}, m) = 1\}.$$

As such, we use the OP to measure the difference between the empirical order derived from the ground truth and the numerical MTE order calculated by the metric m as follows:

$$OP(m) = \frac{|W|}{p}.$$

It is easy to see that the value of OP ranges from 0 to 1. The higher the OP is, the smaller the difference between the ground truth order and the MTE effectiveness order. If a test effectiveness metric has a small difference, it can be regarded as a good metric.

An advantage of OP is it never violates theoretical subsumption facts. For instance, it is known that path coverage subsumes BC. This means that a test that improves the BC score will also improve the **path coverage score** (**PCS**). Consequently, if a triggering test case covers an uncovered branch, it will cover an uncovered path. In other words, if BC(A) > BC(B), we can deduce that PCS(A) > PCS(B). In other words, PCS is deduced to be no worse than BC in terms of OP. This deduction is not captured by other statistical indicators. For example, when using statistical indicators, it is possible that the Spearman's r of BC is bigger than PCS.

4 EXPERIMENTAL SETUP

In this section, we describe in detail the experimental design. First, we report the projects and tools used in our study. Then, we introduce the set of the test suite effectiveness metrics we have investigated. Finally, we list the research questions.

4.1 Experimental Subjects

Projects. We use Defects4J 2.0.0 to compare test effectiveness metrics, which is a benchmark with large real-world Java projects. There are reproducible real faults mined from source code repositories in the benchmark. For each fault, the benchmark contains a buggy version, a fixed version, and at least one triggering test case. We use all the active bugs to compare the metrics. Among all bugs, we filter out those that fail the mutation testing or exceed the time limit (30 minutes). The remaining buggy versions are reported in Table 4.

Mutants. The tool we use to generate and run the mutants is Major [28], which is a widely used tool for Java mutation testing. To generate the entire set of possible mutants, all mutators are used in this article. After the mutants are executed, not only the mutation score can be obtained, but also the detail of which tests are executed can be obtained by modifying the scripts that come with Defects4J. It is worth noting that the mutation testing tool does not support failed test cases. To this concern, all the test criteria including coverage criteria are only calculated in the fixed version.

Tests. For each fixed version, the manually written test cases are used as the whole test suite. We had planned to use an automatic tool to generate test cases. However, we found a serious problem.

Table 4. The Information of Projects in Defects4J

Identifier	Project name	Number of bugs	Active bug ids	Used bug ids
Chart	jfreechart	26	1–26	3–5, 9–11, 13–16, 18–20, 25
Cli	commons-cli	39	1–5, 7–40	1–5, 7–40
Closure	closure-compiler	174	1–62, 64–92, 94–176	61–62, 76, 83, 85, 87, 88, 91, 92, 94, 95, 97–106, 151
Codec	commons-codec	18	1–18	7–18
Collections	commons-collections	4	25–28	–
Compress	commons-compress	47	1–47	1–39
Csv	commons-csv	16	1–16	1–13
Gson	gson	18	1–18	1–13
JacksonCore	jackson-core	26	1–26	1–26
JacksonDatabind	jackson-databind	112	1–112	1–7, 9–16, 18, 20, 28, 29, 34, 35, 39, 42, 43, 45, 46, 49, 62, 71, 72
JacksonXml	Jackson-dataformat-xml	6	1–6	1–6
Jsoup	jsoup	93	1–93	–
JxPath	commons-jxpath	22	1–22	1–22
Lang	commons-lang	64	1, 3–65	1, 3–65
Math	commons-math	106	1–106	1–14
Mockito	mockito	38	1–38	24, 25, 27–38
Time	joda-time	26	1–20, 22–27	1–12, 14–20, 22–27

Our assumption is that the effectiveness of the whole set will not be weaker than its subset. But the automatic tool posed a big threat to this assumption: for example, EvoSuite, which is most commonly used, has a major feature of assuming that the source program behaves correctly and generates assertions based on that. At that point, then, if the current version of the code is buggy, it will also generate assertions based on the wrong behavior. Specifically on Defetcs4J, if we generate test cases on the buggy version: "gen_tests.pl -g evosuite -p Lang -v 1b". The resulting test suite will have test cases that pass on the buggy version and fail on the fixed version. As a result, in RQ4, we generate test cases on the fixed version and check them on the buggy version. Then if some test cases fail on the buggy version, we label them as triggering test cases. Among 361 bugs, we generated 123 automatic test suites with triggering test cases on 123 bugs.

Coverage. The tool we use to collect the coverage information is Cobertura [2], which is a widely used coverage tool. SC and BC can be obtained with the scripts that come with Defects4J.

4.2 Test Suite Effectiveness Metrics

In this study, we compare the accuracy of seven test suite effectiveness metrics: two code coverage metrics (SC and BC, which can be collected by Cobertura) and five mutation score metrics (the original mutation score MS, COS, RMS, SMS, and CMS). In our study, MS is obtained directly by Major.

(1) COS. The 5-Operators Selection proposed by Offutt et al. [36] is the approach chosen in this article. For Major, we use "LVR", "AOR", "ROR", "LOR", and "ORU" among all mutators as the implementation of COS.
(2) RMS. To implement RMS, given the ratio of selected mutants n%, we keep randomly selecting one of the remaining mutants with equal probability until n% of all mutants are selected. In this article, n = 30.
(3) SMS. To select the subsuming mutants used for computing mutation scores, we use the matrix which records all test cases that kill a mutant. Then we select all the mutants, which are not subsumed by other mutants, to compute a mutation score as the metric.
(4) CMS. To implement CMS, we need to collect feature data for each mutant. In this article, an instance for a mutant is a 0-1 list based on the killing matrix. For example, the instance of m is [1, 0, 1, 0], which means that the first and third test case kills m and the other two test cases do not kill m. Then, the k-means algorithm is applied to all mutant instances to classify the mutants into different clusters. Here, we set the "k" to be equal to the number

of selected mutants by SMS. After that, the selection is obtained by randomly selecting one mutant from each cluster.

4.3 Research Question

In this article, we propose the following questions to evaluate the investigated metrics:

RQ1: Based on ASSENT, which MTE is the most accurate metric under the ground truth criteria?

The answer to RQ1 can find the best metric for evaluating test suite effectiveness. To investigate RQ1, for an MTE m, we compute the OP(m) for a project with all the used bugs. For example, there are 6 bugs for the project CSV. Therefore, we can obtain 6 test suite pairs. In a pair, one test suite is the whole test suite and the other is its maximal subset without triggering test cases. According to the ground truth, the former one is more effective than the latter. For a metric, we check whether the ground truth of the 6 test suite pairs is satisfied and compute the OP for this metric. The larger the OP value, the more accurate the metric m is. Considering that there is randomness in 2 metrics (i.e., RMS and CMS), we will repeat each technology 20 times to obtain an average OP value.

For RQ1, we use the following, ASSENT configuration:

— **Ground Truth:** A is more effective than B iff 1. B is A's subset and 2. A can detect more real fault than B.
— **Benchmark Test Suites:** A = T, B = T-C. Here, T is the whole test suite. C is the triggering test case set.
— **Agreement Indicator: OP.**

RQ2: Based on ASSENT, will the use of mutants instead of real faults lead to overestimation or underestimation of the MTE?

In existing studies, the use of mutants instead of real defects is often a threat. However, it is unknown that whether it will lead to overestimation or underestimation of the MTE. To investigate RQ2, we use the alternative ground truth (in Section 3.1.2) to reevaluate the MTEs. In other words, OP is the proxy of the agreement between MS and an MTE. After that, we compare the results to RQ1. Finally, we can quantitatively analyze the differences between the two RQs.

For RQ2, we use the following, ASSENT configuration:

— **Ground Truth:** A is more effective than B iff 1. B is A's subset and 2. A can detect (i.e., kill) more mutants than B.
— **Benchmark Test Suites:** A = T, B = T-C. Here, T is the whole test suite. C is the triggering test case set.
— **Agreement Indicator: OP.**

RQ3: Based on ASSENT, will the size of test suite influence the conclusion?

Would the conclusions change if the size is controlled, as in the existing work? In other words, we re-implement the experiment by ASSENT by the following configuration:

— **Ground Truth:** For the test suites with the same size, those which can detect the real fault (A) are more effective than the others (B).
— **Benchmark Test Suites:** A = T-NC, B = T-C. Here, T is the whole test suite. C is the triggering test case set and NC is a random non- triggering test cases set with the same size to C.
— **Agreement Indicator: OP.**

RQ4: Based on automatic test generation, which MTE is the most accurate metric under the ground truth criteria?

We first run EvoSuite on the fixed version. Then if there is no triggering test case in the generated pool, we will skip this bug. Using the "branch coverage criteria", we have generated test suites with at least one triggering test case on 123 bugs. The corresponding 123 automatic test suites can be found in our open repository. Finally, we implement the experiment by ASSENT by the following configuration:

— **Ground Truth:** A is more effective than B iff 1. B is A's subset and A can detect more real faults than B.
— **Benchmark Test Suites:** A = T, B = T-C. Here, T is the whole automatic test suite. C is the triggering test case set.
— **Agreement Indicator: OP.**

RQ5: Based on ASSENT, which MTE is the most accurate metric following the setup in [39]?

In order to implement a complete comparison with the previous work, the setup under this RQ, follows the setup of [39]:

— **Ground Truth:** For the test suites with the same size, those which can detect the real fault (A) are more effective than the others (B).
— **Benchmark Test Suites:** A = Random(T, r%), B = Random(T, r%). Here, T is the whole test suite. r% is the sampling ratio. Random(T, r) obtained by sampling r% in T. In our article, r = 10,20,30,40,50, and we sample each ratio 100 times.
— **Agreement Indicator:** Kendall, Spearman, Pearson.

RQ6: By probabilistic coupling, which MTE is the most accurate metric? In [10], **probabilistic coupling** (**PC**) was employed to assess an MTE's capability in detecting actual faults. PC is defined as follows: For an MTE's test goal g (e.g., to kill a mutant) and a real fault f, $PC(f, g) = P(\text{detect } f | g \text{ is detected})$, which represents the probability of detecting the real fault when selecting a test that satisfies the test goal. When considering the MTE's entire test goals G (e.g., killing a set of mutants) and a real fault f, $PC(f, G)$ is computed as the maximum value of $PC(f, g)$ for all g in G: $PC(f, G) = \max_{g \in G} PC(f, g)$.

It is important to note the following characteristics of this metric: (1) PC is limited in its computational applicability to a single fault as the ground truth. This makes it less directly applicable when the ground truth comprises multiple targets. Consequently, this article only utilizes PC as originally defined in [10]; (2) Ideally, PC should be calculated using the entire kill/coverage matrix without any form of sampling. This is because employing all the test cases will yield the most accurate estimation of probabilities; (3) PC is not directly applicable for BC since a BC goal may be 50% achieved.

In order to implement a complete comparison with the previous work, the setup under this RQ are:

— **Ground Truth:** for single test case A and B, A is more effective than B iff and A can detect the real fault while B cannot.
— **Benchmark Test Suites:** A = one triggering test case which can hit the test goal g. B= one non-triggering test case which can hit the test goal g.
— **Agreement Indicator:** PC(f, g) = |A|/(|A| + |B|).

5 EXPERIMENTAL RESULTS

5.1 Evaluation of MTEs based on ASSENT

Table 5 shows the average OP values of 7 metrics for 15 projects. Before analyzing the results, we need to reiterate that our focus is on the agreement between a metric and a ground truth criterion.

Table 5. (RQ1) The Average OP of Seven Metrics on 15 Projects

Project	MS	COS	RMS	SMS	CMS	SC	BC
Chart	0.889	0.778	0.691	0.889	0.589	0.611	0.611
Cli	0.718	0.487	0.523	0.718	0.290	0.487	0.641
Closure	0.885	0.654	0.581	0.885	0.326	0.423	0.577
Codec	0.833	0.750	0.708	0.833	0.488	0.416	0.583
Compress	0.846	0.718	0.784	0.846	0.381	0.564	0.564
Csv	0.923	0.538	0.369	0.923	0.527	0.385	0.615
Gson	0.769	0.462	0.503	0.769	0.404	0.692	0.769
Jacksoncore	0.885	0.846	0.787	0.885	0.521	0.808	0.808
JacksonDatabind	0.800	0.567	0.450	0.800	0.300	0.733	0.733
JacksonXml	0.833	0.667	0.633	0.833	0.200	0.667	0.667
Jxpath	0.773	0.636	0.545	0.773	0.452	0.364	0.500
Lang	0.922	0.859	0.749	0.922	0.618	0.750	0.813
Math	0.571	0.428	0.418	0.571	0.443	0.500	0.357
Mockito	0.615	0.462	0.335	0.615	0.408	0.231	0.462
Time	0.760	0.720	0.618	0.760	0.410	0.360	0.440
avg.	0.801	0.638	0.580	0.801	0.424	0.533	0.609

As a result, we may not conclude that a metric is more practical than another metric by OP. RMS, for instance, can set different ratios for selection. If we assume that RMS(n%) selects the n% of the mutants. We would intuitively believe that the larger the selection is, the smaller the difference between the metric and the ground truth criterion is, and the larger the executing cost is. Due to this, it is unreasonable to conclude that RMS(90%) is more practical than RMS(30%) by OP. What we can conclude is that RMS(90%) can provide a more accurate metric for evaluating the effectiveness of test suites than RMS(30%). In order to facilitate readers to have an in-depth comparison of these metrics, we list their costs by the number of executed mutants. For SC and BC, none of the mutants need to be executed. For MS, SMS, and CMS, all of the mutants need to be executed. For COS, more than 30% of the mutants need to be executed on most programs. For RMS, 30% of the mutants need to be executed. However, since we have to repeat RMS for 20 times, all the mutants are executed in the experiment.

From Table 5, we have the following observations:

— MS and SMS are the most accurate metrics among the 7 metrics. On average, 80% of the enhancement of the test suite effectiveness will be expressed by MS or SMS. In other words, if we add triggering tests to detect an exposed fault, there is an 80% chance that the mutation score will increase. This observation is consistent with [27]. In their research, the OP for MS is 75%.

— The performance of SMS is the same as that of MS. First, we explain why SMS must increase if MS increases. If a triggering test case leads to an increase in mutation score, it must kill a mutant which is not killed by other test cases. As a result, this mutant must be a subsuming mutant. Then, this mutant will be selected by SMS. Finally, this mutant leads to the increase of SMS. Second, we explain why SMS will not increase if MS does not increase. If a triggering test case does not lead to an increase in mutation score, it means that the test suites A and B are not distinguished by the whole mutant set. For a subset of the whole set, it must not distinguish between A and B. Based on the two explanations, we can conclude that the performance of SMS is the same as that of MS.

Table 6. The p-value (above) Computed by the Wilcoxon Signed Rank Test for 4 Metrics After Benjamini-Hochberg Adjustment and the Cliff's Delta δ (below) for 4 Metrics

effect size p	**COS**	**RMS**	**SC**	**BC**
COS	–	**0.042**	0.086	0.505
RMS	**−0.227 (small)**	–	0.495	0.561
SC	**−0.364 (medium)**	**−0.173 (small)**	–	0.075
BC	−0.106	0.111	**0.271 (small)**	–

$p < 0.05$ indicates there is a significant difference between the two corresponding metrics (i.e., bold value). $|\delta| \geq 0.147$ indicates a nontrivial effective size (i.e., bold value).

— The common reduction strategies (COS, RMS) will cause a significant decline in OP (even the performance of BC and SC is comparable to them). To summarize, COS, RMS, CMS, BC, and SC may be risky and insensitive to evaluate test suite effectiveness. By adding triggering test cases, the effectiveness of test suits increases. However, such a fact cannot be accurately reflected by these metrics.

Considering that the performance of COS, RMS, SC, and BC are close, we conduct a hypothesis test on whether there are significant differences between the four metrics by the Wilcoxon signed rank test. The p-value after Benjamini-Hochberg adjustment is shown in Table 6. To measure the magnitude of the difference, we employ the Cliff's delta δ, which is used for median comparison on effect size, to examine whether the magnitude of the difference is practically important. The delta is shown in Table 6. From Table 6, to summarize the results of p and delta, by OP, it is hard to conclude that COS is better than BC.

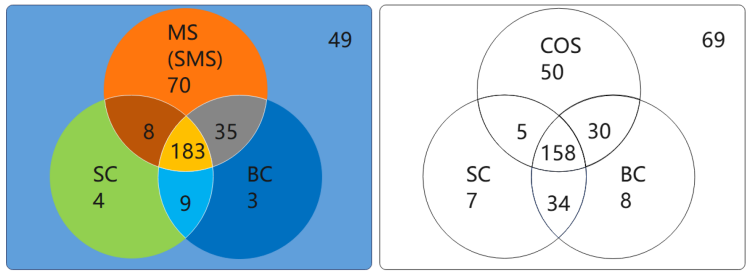
After comparing the OP values of several metrics, we also want to investigate the overlapping relationship between the maintained ground truth order. When the order among test suite pairs is not maintained by an MTE, it means that this MTE does not take the corresponding fault into consideration, i.e., adding or deleting the triggering test cases will not change the metric value. In the following, if this happens, we can say this fault is not considered by this MTE.

Figure 2(a) shows the Venn diagram for all the 361 faults. The number of faults considered by a MTE is attached in the corresponding circle. The overlapping part represents the faults considered by multiple metrics. Considering that there is randomness in RMS and CMS, we do not draw the diagram for them. In addition, another interesting question is the relationship between the types of faults[7] against MTEs. Figure 2(b) and (c) shows the relationship.

From the figure, we have the following observations:

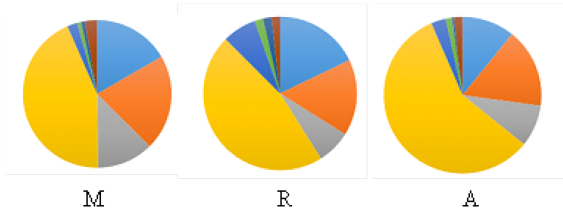
— By all the metrics, 49 out of 361 faults are not considered. In other words, for these 49 faults, when testers add test cases into an existing test suite to detect them, none of the metric values will increase. This observation hints that there is a large space for improving the existing metrics or proposing new ones.
— When comparing MS to BC and SC, most faults are considered by all of them. Less than 5% ((4 + 9 + 3)/361) of faults are considered by coverage metrics alone. To this concern, their effectiveness is lower than MS.
— When comparing COS to BC and SC, coverage metrics are also competitive. COS reflects 50 unique faults while coverage metrics reflect 49 (i.e., 7 + 8 + 34). As a result, it is unreasonable to use COS instead of MS to compare coverage metrics. This observation points out an empirical threat to the findings of [17].

[7]Here we refer to [43] for the types of faults. We divide the faults into three categories according to the patches: M(fixed by modifying), A(fixed by adding) and R(fixed by removing).



(a) the fault coupling between the MTEs

(b) the fault types



(c) the visualization of the corresponding ratio in (a).

Fig. 2. The diagram to show (a) the fault coupling between the MTEs, (b) the fault types, and (c) the visualization of the corresponding ratio. See footnote 7 for the fault types.

— Most of the faults are fixed by modificatins(M) and additions(A) while few defects are fixed by removal(R).
— By the pie charts, we can see that the percentage of yellow in "A" is the highest. This indicates that if a fault is fixed by adding code, it is more likely to be considered by MTE. This observation is easily explained by the fact that the added test cases (i.e., triggering test cases) are most likely customized for the newly added code fragment. It is normal then to cause an increase in MTE values by these test cases.

To conclude, MS and SMS provide the most accurate metric among the 7 metrics under the proposed ground truth.

5.2 The Proposed Ground Truth v.s. the Alternative Ground Truth

Table 7 shows the average OP values of 6 metrics on 15 projects under the alternative ground truth. The data in brackets indicates how much it has changed compared with RQ1. For example, the "+14%" in the first cell is computed by (0.889-0.778)/0.778.

Table 7. (RQ2) The Average OP of Six Metrics on 15 Projects Under Alternative Ground Truth with Comparison to RQ1

Project	COS	RMS	SMS	CMS	SC	BC
Chart	0.889(+14%)	0.803(+16%)	1.000(+12%)	0.700(+18%)	0.722(+18%)	0.722(+18%)
Cli	0.770(+58%)	0.805(+54%)	1.000(+39%)	0.572(+97%)	0.615(+26%)	0.718(+12%)
Closure	0.770(+18%)	0.697(+20%)	1.000(+13%)	0.443(+35%)	0.539(+27%)	0.615(+6%)
Codec	0.917(+22%)	0.875(+24%)	1.000(+20%)	0.655(+34%)	0.593(+43%)	0.750(+28%)
Compress	0.872(+21%)	0.812(+4%)	1.000(+18%)	0.535(+40%)	0.615(+9%)	0.615(+9%)
Csv	0.615(+14%)	0.446(+21%)	1.000(+8%)	0.704(+34%)	0.308(-20%)	0.693(+13%)
Gson	0.693(+50%)	0.735(+46%)	1.000(+30%)	0.635(+57%)	0.770(+11%)	0.857(+11%)
Jacksoncore	0.961(+14%)	0.902(+15%)	1.000(+13%)	0.637(+22%)	0.770(-5%)	0.846(+5%)
JacksonDatabind	0.767(+35%)	0.650(+44%)	1.000(+25%)	0.500(+67%)	0.800(+9%)	0.867(+18%)
JacksonXml	0.834(+25%)	0.800(+26%)	1.000(+20%)	0.367(+84%)	0.834(+25%)	0.834(+25%)
Jxpath	0.864(+36%)	0.773(+70%)	1.000(+29%)	0.680(+50%)	0.500(+37%)	0.637(+27%)
Lang	0.937(+9%)	0.837(+12%)	1.000(+8%)	0.696(+13%)	0.828(+10%)	0.891(+10%)
Math	0.853(+99%)	0.846(+103%)	1.000(+75%)	0.871(+96%)	0.929(+86%)	0.786(+120%)
Mockito	0.846(+83%)	0.720(+115%)	1.000(+62%)	0.793(+94%)	0.615(+160%)	0.846(+83%)
Time	0.996(+38%)	0.858(+38%)	1.000(+31%)	0.650(+58%)	0.520(+44%)	0.680(+54%)
avg.	0.839(+32%)	0.771(+33%)	1.000(+24%)	0.630(+48%)	0.664(+24%)	0.757(+24%)

The data in brackets indicates how much it has changed compared with RQ1.

Table 8. (RQ2) The p-value (above) Computed by the Wilcoxon Signed Rank Test for 6 Metrics' Change Rates After Benjamini-Hochberg Adjustment and the Cliff's Delta (Below) for 6 Metrics' Change Rates

effect size \p	COS	RMS	SMS	CMS	SC	BC
COS	–	0.218	**0.005**	**0.005**	0.474	0.372
RMS	0.084	–	**0.015**	0.094	0.382	0.186
SMS	**−0.236 (small)**	**−0.240 (small)**	–	**0.005**	1	0.999
CMS	**0.356 (medium)**	**0.280 (small)**	**0.596 (large)**	–	0.060	**0.018**
SC	**−0.182 (small)**	**−0.244 (small)**	−0.04	**−0.489 (large)**	–	0.877
BC	**−0.311 (small)**	**−0.298 (small)**	**−0.173 (small)**	**−0.587 (large)**	−0.031	–

($p < 0.05$) indicates there are significant differences between the two corresponding metrics (i.e., bold value).
($|\delta| > 0.147$) indicates a nontrivial effect size (i.e., bold value).

From the table, we have the following observations:

— In most cases, 6 metrics are overestimated by the alternative ground truth. Only SC may be underestimated in few cases. On average, all the metrics are overestimated by more than 20%.

— Mutation score metrics are more likely to be overestimated than code coverage metrics. In fact, if all the metrics are overestimated by the same ratio, the rank for the metrics will be the same. As a result, we need to analyze the influence of overestimation for the rank. In Table 5, the rank for COS, RMS, SMS, CMS, SC, and BC is 2, 4, 1, 6, 5, and 3 while in Table 7, the rank is 2, 3, 1, 6, 5, and 4. The rank of BC changed most. As a result, it is not suitable to compare mutation score metrics to code coverage metrics under alternative ground truth. However, if we just compare mutation score metrics or code coverage metrics, the rank for the MTEs is consistent to the rank in RQ1.

Furthermore, for the change rate, we conduct a hypothesis test on whether there are differences between the 6 metrics' change rates by the Wilcoxon signed rank test. To measure the magnitude of the difference, the Cliff's delta is shown in Table 8. From Table 8, to summarize the results of p and delta, when using the alternative ground truth, CMS will be overestimated more than the other

Table 9. (RQ3) The Average OP of Seven Metrics on 15 Projects Under Size Control

Project	MS	COS	RMS	SMS	CMS	SC	BC
Chart	0.714	0.573	0.536	0.659	0.464	0.483	0.505
Cli	0.650	0.425	0.367	0.500	0.200	0.444	0.517
Closure	0.647	0.546	0.454	0.597	0.321	0.215	0.286
Codec	0.769	0.685	0.612	0.681	0.319	0.376	0.574
Compress	0.820	0.688	0.628	0.777	0.362	0.580	0.580
Csv	0.821	0.475	0.317	0.812	0.525	0.348	0.578
Gson	0.919	0.535	0.592	0.919	0.438	0.644	0.681
Jacksoncore	0.828	0.802	0.702	0.686	0.450	0.775	0.743
JacksonDatabind	0.288	0.225	0.191	0.288	0.103	0.218	0.352
JacksonXml	0.825	0.608	0.650	0.816	0.341	0.633	0.685
Jxpath	0.603	0.458	0.396	0.608	0.369	0.227	0.492
Lang	0.775	0.726	0.661	0.656	0.438	0.559	0.639
Math	0.617	0.526	0.475	0.528	0.356	0.428	0.451
Mockito	0.650	0.283	0.333	0.650	0.450	0.382	0.452
Time	0.647	0.647	0.561	0.641	0.300	0.278	0.429
avg.	0.705	0.547	0.498	0.655	0.362	0.439	0.531

metrics. To conclude, on average, all the metrics are overestimated by more than 20% under the alternative ground truth. Meanwhile, mutation score metrics are more likely to be overestimated than code coverage metrics on average. As a result, it is not suitable to compare mutation score metrics with code coverage metrics under the alternative ground truth.

5.3 Removing Triggering v.s. Removing Random

The scenarios designed in RQ3 have been adopted more widely in existing studies: assuming that the set of test cases that can detect defects is more effective under the premise of controlling the size, how would the conclusions change? It can be expected that we will not observe an increase in correlation. Therefore, we are more concerned with the magnitude of the decrease in observation. In addition, there are two ways to interpret the conclusions: one is this means the true correlation after removing the mixed effects of scale; the other is there is a certain risk in this ground truth, as illustrated by examples of Chart 3, 9, and 17 in the introduction. When we base our evaluation on different versions, we may make different evaluations of the effectiveness on t3, t9, and t17.

The results are in Table 9. From the table, we have the following observations:

— OP values for all MTEs have dropped significantly. Except MS, all other MTEs have a significant inconsistency with the ground truth (i.e., OP < 0.7). In other words, if we believe that the triggering cases are more effective than the other cases, most of the existing metrics do not highly agree with this assumption: the contribution of the other test cases to calculating the MTE values is likely to be higher than the triggering cases.
— Our conclusions on comparing several MTEs changed little. The average ranks of MS, COS, RMS, CMS, SC, and BC are the same as RQ1. To summarize, the average comparison of MTEs under this ground truth is concluded to be consistent with RQ1.

To conclude, MS provides the most consistent metric among the 7 metrics. However, by MS, although all the mutant is considered, on average, there is a 30% (i.e., 1-0.705) probability that triggering test cases cannot kill more unique mutants than the non-triggering test cases. But we

Table 10. (RQ4) The Average OP of Seven Metrics by Automatic Tests

Project	MS	COS	RMS	SMS	CMS	SC	BC
Chart	1.000	1.000	0.924	1.000	0.905	0.895	0.895
Cli	0.929	0.786	0.804	0.929	0.829	0.857	0.929
Closure	1.000	1.000	0.838	1.000	0.750	0.500	1.000
Compress	1.000	0.941	0.788	1.000	0.803	0.941	0.941
Csv	0.833	0.500	0.475	0.833	0.542	1.000	1.000
Gson	1.000	1.000	0.788	1.000	0.863	0.750	1.000
JacksonCore	1.000	0.667	0.689	1.000	0.800	0.556	0.889
JacksonDatabind	1.000	0.667	0.450	1.000	0.667	1.000	1.000
JacksonXml	1.000	0.500	0.800	1.000	1.000	1.000	1.000
JxPath	1.000	0.750	0.633	1.000	0.821	0.750	1.000
Lang	0.923	0.808	0.792	0.923	0.775	0.846	0.962
Math	0.800	0.800	0.700	0.800	0.790	1.000	0.800
Mockito	1.000	0.500	0.450	1.000	1.000	0.500	1.000
Time	1.000	0.875	0.719	1.000	0.656	0.875	1.000
avg.	0.963	0.771	0.703	0.963	0.800	0.819	0.958

remain skeptical about the ground truth. The label of triggering test cases is likely to change with version. As a result, the assumption that a triggering test case is more effective than a non-triggering test case is a much less reliable one.

5.4 Manual Test Suites v.s. Automatic Test Suites

This subsection compares manual test suites with automatic test suites. However, in Table 10, only automatically generated test cases were considered for the total pool. This comparison is achieved by comparing the experimental results with the results of RQ1. Since there is heterogeneity between test cases generated by different technologies, we do not mix all test cases to form a total test case pool. By comparing two results, the readers can better understand the generality in conclusions and the heterogeneity between manual and automatic.

From Table 10, comparing the conclusions from RQ1, we have identified both similarities and differences. The similarities lie in the fact that mutation score MS and SMS remain the most effective metrics, and that BC proves to be superior to SC. However, there is a notable difference in the OP for coverage metrics, which is notably better even when compared to most reduced mutation scores.

Upon analyzing the reasons behind this difference, we find the following insights:

— EvoSuite generates test cases with coverage as a fitness function, and even when using "weak mutation criteria", coverage remains an important factor. After test case generation, EvoSuite applies test reduction, resulting in a test suite where each test case significantly contributes to (branch) coverage. Deleting any of these test cases would lead to a decrease in coverage, thereby preserving the order relationship between A and B.

— It is probable that programs that have a detrimental effect on coverage are filtered out. To obtain a pair of (A, B), EvoSuite must generate triggering test cases. This implies that if EvoSuite fails to generate triggering test cases for specific programs, it is likely that the effectiveness of coverage metrics was weak on them. However, these programs have been filtered out. As we noted, 123 out of 361 bugs are triggered by EvoSuite. From this perspective, the true validity of BC is probably much lower.

Ratio	MS	COS	RMS	SMS	CMS	SC	BC
10%	0.146	0.148	0.146	0.159	0.150	0.139	0.141
20%	0.162	0.166	0.158	0.164	0.150	0.146	0.147
30%	0.158	0.155	0.138	0.168	0.136	0.149	0.155
40%	0.176	0.169	0.154	0.179	0.147	0.155	0.154
50%	0.189	0.180	0.172	0.181	0.139	0.157	0.162
avg.	0.166	0.164	0.154	0.170	0.144	0.149	0.152

Fig. 3. (RQ5) The average Kendall's Tau on all used bugs of seven metrics by 10%–50% sampling ratio.

Ratio	MS	COS	RMS	SMS	CMS	SC	BC
10%	0.172	0.171	0.168	0.172	0.166	0.163	0.165
20%	0.192	0.194	0.183	0.179	0.165	0.171	0.179
30%	0.188	0.181	0.162	0.187	0.149	0.175	0.185
40%	0.210	0.196	0.178	0.199	0.161	0.180	0.184
50%	0.221	0.206	0.195	0.200	0.152	0.180	0.185
avg.	0.197	0.190	0.177	0.187	0.159	0.174	0.180

Fig. 4. (RQ5) The average Spearman's ρ on all used bugs of seven metrics by 10%-50% sampling ratio.

Additionally, we want to emphasize the conservative nature of our study's conclusions: the OP we examined for the test case pairs indicates the percentage of preservation for each metric. However, for the unexamined test case pairs, the results remain unknown. Unlike a general comparison for all bugs was made in RQ1, for RQ4, we have to filter out bugs where coverage may not perform well. This is done to ensure that EvoSuite could generate triggering test cases successfully.

To summarize, we confirmed that SMS and MS are still the best metrics. Overall, these findings highlight the continued effectiveness of SMS and MS as metrics, and the importance of considering the specific characteristics and limitations of coverage metrics when analyzing their performance.

5.5 OP v.s. Statistical Agreement Indicators

Figures 3–5 present the average statistical indicators of 7 metrics at different sampling ratios. The following observations can be made:

— Generally, most metrics exhibit similar statistical indicators, indicating that the differences between MTEs are not significant. However, OP stands out as significantly better at distinguishing.
— All MTEs show only a low correlation with the ground truth, which aligns with previous studies conducted under the same setups. Meanwhile, this supports our theoretical analysis in Section 3.3. Therefore, relying solely on statistical indicators is not suitable when using real defects as the ground truth, as they may not accurately reflect the effectiveness of MTE.
— In terms of statistical indicators, SMS and MS remain the best MTEs. Furthermore, a trend can be observed: the correlation of MTE tends to increase with higher sampling ratios in most cases.

Ratio	MS	COS	RMS	SMS	CMS	SC	BC
10%	0.179	0.176	0.172	0.176	0.167	0.161	0.163
20%	0.194	0.193	0.185	0.186	0.167	0.165	0.164
30%	0.188	0.181	0.164	0.192	0.151	0.167	0.172
40%	0.202	0.191	0.177	0.201	0.165	0.171	0.179
50%	0.214	0.202	0.197	0.203	0.154	0.172	0.186
avg.	0.195	0.189	0.179	0.192	0.161	0.167	0.173

Fig. 5. (RQ5) The average Pearson's r on all used bugs of seven metrics by 10%-50% sampling ratio.

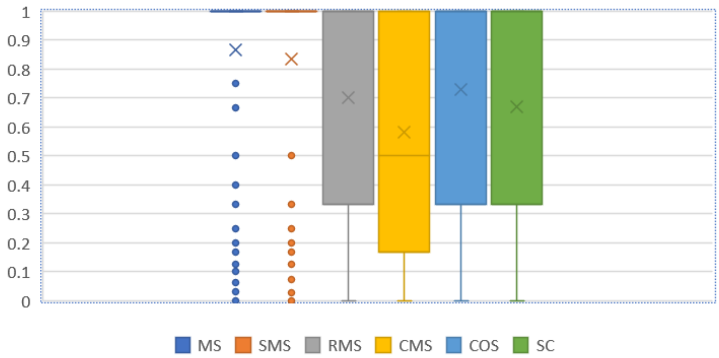


Fig. 6. (RQ6) The boxplot for 6 MTEs on PC.

In conclusion, the results indicate that since there are similarities in the statistical metrics among the MTEs, OP shows a stronger ability to distinguish. Furthermore, the low correlation supports our conclusions in Section 3.3, where we recommend against using statistical indicators as an accurate measure of MTE under real defects. Finally, SMS and MS stand out as the best MTEs overall. Increasing the sampling ratio generally leads to a higher correlation between MTE and the ground truth.

The PC performance of the 6 MTEs is illustrated in Figure 6. The average PC values for MS, SMS, RMS, CMS, COS, and SC are 0.867, 0.834, 0.700, 0.581, 0.729, and 0.670, respectively. According to the PC metric, SMS and MS remain the top-performing MTEs. Importantly, these comparative findings align closely with the conclusions drawn in RQ1: MS and SMS outperform COS, which in turn outperforms RMS, SC, and CMS, highlighting the robustness of our conclusions.

However, it is crucial to acknowledge that the use of PC has inherent limitations and may not be directly applicable in various scenarios. For instance, when dealing with multiple targets as the ground truth (e.g., multiple faults in a version), PC cannot be computed directly. To address such scenarios, the development of extensions and variants of PC becomes necessary.

Therefore, while SMS and MS emerge as the top MTEs based on PC evaluation, it is important to consider the limitations of PC and explore additional approaches when dealing with complex scenarios involving multiple targets as the ground truth.

5.6 Probabilistic Coupling

The PC performance of the 6 MTEs is illustrated in Figure 6. The average PC values for MS, SMS, RMS, CMS, COS, and SC are 0.867, 0.834, 0.700, 0.581, 0.729, and 0.670, respectively. According

to the PC metric, SMS and MS remain the top-performing MTEs. Importantly, these comparative findings align closely with the conclusions drawn in RQ1: MS and SMS outperform COS, which in turn outperforms RMS, SC, and CMS, highlighting the robustness of our conclusions.

However, it is crucial to acknowledge that the use of PC has inherent limitations and may not be directly applicable in various scenarios. For instance, when dealing with multiple targets as the ground truth (e.g., multiple faults in a version), PC cannot be computed directly. To address such scenarios, the development of extensions and variants of PC becomes necessary.

Therefore, while SMS and MS emerge as the top MTEs based on PC evaluation, it is important to consider the limitations of PC and explore additional approaches when dealing with complex scenarios involving multiple targets as the ground truth.

6 RECOMMENDATIONS AND IMPLICATIONS

Recommendations. We mainly provide two recommendations:

- The first is, if a study tries to evaluate a test suite effectiveness metric, please use ASSENT as the guideline. First, there is a need to define the ground truth. Then, test suite pairs for comparison should be generated based on the ground truth. Finally, the agreement indicator should be determined to evaluate the metric. The benefit is twofold. On the one hand, it is easy to compare the experimental setups with other related work. On the other hand, it is clear for readers to understand the work and find out the potential risks.
- The other recommendation is that when researchers try to quote or refute some conclusions (e.g., one wants to quote that there is a strong correlation between mutation detection ratio and real fault), they cannot simply pick the conclusions of some literature and quote them directly. Instead, they must pay careful attention to whether the three elements that produce the conclusion are applicable in their scenario. If not, for example, some ground truths cannot be widely recognized in their scenario, they should not quote that conclusion.

Implications. Our study has important implications for both researchers and practitioners. The detailed implications are listed as follows:

- We provide a framework that can help researchers to investigate the accuracy of the test effectiveness metrics effectively. Meanwhile, it is easy to compare the experimental setups in the related work by our framework.
- We analyze the possible threat caused by representative ground truths. On the one hand, the threat of "real fault" is analyzed in Section 3.1.1. On the other hand, the threat of "mutant" is analyzed in RQ2. When the researchers are going to cite the corresponding conclusions, to eliminate the threat, they must pay attention to whether the three elements are consistent with the cited article or not. In addition, they should also explain how the pitfalls we point out will influence their work.
- For the practitioners, we help them to rebelieve that using MS or SMS is still the least risky metric among the MTEs. Using other metrics, there will be a higher risk of ignoring the enhancement of test suite effectiveness.

7 THREATS TO VALIDITY

In this section, we discuss the important threats to the construct validity, internal validity, and external validity of our study. Construct validity denotes the extent to which the variables used in our study accurately measure what we purport to measure. Internal validity is the degree to which conclusions can be drawn about the causal effect of independent variables on the dependent variable. External validity is the degree to which the results of the research can be generalized to the population under study and other research settings.

7.1 Construct Validity

In our study, we use OP to measure consistence between ground truth and MTEs. As a result, the threat comes from the proposed ground truth. As we adopt a conservative ground truth, its advantage is easy to be recognized by researchers, which makes our conclusion conservative. In other words, an MTE with a low OP is useless, while an MTE with a high OP may not be useful. For example, if we use the size of test suites as a metric, by our ground truth, it will achieve the 1.0 (i.e., highest) OP. As a result, we cannot deny that using the best metric under our ground truth may be risky in other cases. However, the risk of an MTE revealed by our study is objective. For an MTE with a low OP, it cannot be sensitively aware of the improvement of the test suite effectiveness. As a result, it has a low ability to evaluate test suite effectiveness. Besides, to minimize the influence of size, we use the whole set as A and filtering triggering test cases as B, which makes the size as close as possible.

7.2 Internal Validity

The main threat is the selection bias of the MTEs. In order to reduce this threat, the seven metrics we compared should be as representative as possible. On the one hand, metrics for Mutation Testing are mostly included. On the other hand, for coverage measurement, we have selected validated and highly effective BC [17] and widely used SC. In the future, we will investigate more metrics.

7.3 External Validity

The main threat is that our evaluation is performed on a limited set of real faults. The distribution of real faults in the real world may be different from the distribution in the limited set. To partially mitigate this threat, we conducted an extended experiment (RQ2) using simulated defects and obtained similar conclusions. To further mitigate this threat, we plan to replicate our study across more programs, test suites, and mutation tools in the future.

8 CONCLUSION

This article provides guidelines for evaluating any metric on the effectiveness of test suites. After revisiting the existing work, we propose a framework ASSENT for evaluating the accuracy of any test suite effectiveness metric. Under this framework, the accuracy of test suite effectiveness metrics can be consistently and accurately evaluated and compared. On the one hand, ASSENT can help researchers to investigate the test metrics effectively. On the other hand, ASSENT can help readers to understand the thread in existing work.

An experiment can be designed with ASSENT easily by addressing the following three questions:

— When can we say that test suite A is more effective than test suite B? (ground truth)
— How to generate A and B? (benchmark test suites)
— Which indicator should be chosen to report the agreement between the MTE and the ground truth? (agreement indicator)

Then we apply this framework to compare representative test effectiveness metrics. As a result, we find that: based on the real faults, mutation score and SMS are the best metrics; if we use mutants as the alternative ground truth, MTEs will be overestimated by more than 20% in values. In the future, we strongly suggest that any newly proposed test effectiveness metrics should be evaluated under ASSENT to demonstrate the practical value.

ACKNOWLEDGMENTS

We would like to thank Kui Liu and Xin Xia for discussions about this work.

REFERENCES

[1] [n. d.]. https://github.com/zhangpengNJU/ASSENT/README.md. Accessed on: 2023-09-25.

[2] [n. d.]. https://github.com/cobertura. Accessed on: 2023-09-25.

[3] Iftekhar Ahmed, Rahul Gopinath, Caius Brindescu, Alex Groce, and Carlos Jensen. 2016. Can testedness be effectively measured?. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. 547–558.

[4] Paul Ammann, Marcio Eduardo Delamaro, and Jeff Offutt. 2014. Establishing theoretical minimal sets of mutants. In *Proceedings of the 2014 IEEE 7th International Conference on Software Testing, Verification, and Validation*. IEEE, 21–30.

[5] Thomas Ball. 2004. A theory of predicate-complete test coverage and generation. In *Proceedings of the International Symposium on Formal Methods for Components and Objects*. Springer, 1–22.

[6] Benoit Baudry, Franck Fleurey, and Yves Le Traon. 2006. Improving test suites for efficient fault localization. In *Proceedings of the 28th International Conference on Software Engineering*. 82–91.

[7] Xia Cai. 2006. *Coverage-based Testing Strategies and Reliability Modeling for Fault-tolerant Software Systems*. The Chinese University of Hong Kong (People's Republic of China).

[8] Xia Cai and Michael R. Lyu. 2005. The effect of code coverage on fault detection under different testing profiles. In *Proceedings of the 1st International Workshop on Advances in Model-based Testing*. 1–7.

[9] 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 2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE '17)*. IEEE, 597–608.

[10] Yiqun T. Chen, Rahul Gopinath, Anita Tadakamalla, Michael D. Ernst, Reid Holmes, Gordon Fraser, Paul Ammann, and René Just. 2020. Revisiting the relationship between fault detection, test adequacy criteria, and test set size. In *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*. 237–249.

[11] Trishul M. Chilimbi, Ben Liblit, Krishna Mehra, Aditya V. Nori, and Kapil Vaswani. 2009. Holmes: Effective statistical debugging via efficient path profiling. In *Proceedings of the 2009 IEEE 31st International Conference on Software Engineering*. IEEE, 34–44.

[12] Jacob Cohen. 1992. Quantitative methods in psychology: A power primer. *Psychol. Bull.* 112, 1 (1992), 1155–1159.

[13] Marcio Eduardo Delamaro, Lin Deng, Vinicius Humberto Serapilha Durelli, Nan Li, and Jeff Offutt. 2014. Experimental evaluation of SDL and one-op mutation for C. In *Proceedings of the 2014 IEEE 7th International Conference on Software Testing, Verification, and Validation*. IEEE, 203–212.

[14] Marcio Eduardo Delamaro, Jeff Offutt, and Paul Ammann. 2014. Designing deletion mutation operators. In *Proceedings of the 2014 IEEE 7th International Conference on Software Testing, Verification, and Validation*. IEEE, 11–20.

[15] Pedro Delgado-Pérez, Sergio Segura, and Inmaculada Medina-Bulo. 2017. Assessment of C++ object-oriented mutation operators: A selective mutation approach. *Software Testing, Verification and Reliability* 27, 4-5 (2017), e1630.

[16] Richard A. DeMillo, Richard J. Lipton, and Frederick G. Sayward. 1978. Hints on test data selection: Help for the practicing programmer. *Computer* 11, 4 (1978), 34–41.

[17] 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 Transactions on Software Engineering and Methodology* 24, 4 (2015), 1–33.

[18] Dunwei Gong, Gongjie Zhang, Xiangjuan Yao, and Fanlin Meng. 2017. Mutant reduction based on dominance relation for weak mutation testing. *Information and Software Technology* 81, C (2017), 82–96.

[19] Rahul Gopinath, Carlos Jensen, and Alex Groce. 2014. Code coverage for suite evaluation by developers. In *Proceedings of the 36th International Conference on Software Engineering*. 72–82.

[20] Giovanni Grano, Fabio Palomba, and Harald C. Gall. 2019. Lightweight assessment of test-case effectiveness using source-code-quality indicators. *IEEE Transactions on Software Engineering* 47, 4 (2019), 758–774.

[21] Richard G. Hamlet. 1977. Testing programs with the aid of a compiler. *IEEE Transactions on Software Engineering* 4 (1977), 279–290.

[22] Farah Hariri, August Shi, Vimuth Fernando, Suleman Mahmood, and Darko Marinov. 2019. Comparing mutation testing at the levels of source code and compiler intermediate representation. In *Proceedings of the 2019 12th IEEE Conference on Software Testing, Validation and Verification (ICST '19)*. IEEE, 114–124.

[23] Shamaila Hussain. 2008. Mutation clustering. *Ms. Th., Kings College London, Strand, London* (2008), 9.

[24] Laura Inozemtseva and Reid Holmes. 2014. Coverage is not strongly correlated with test suite effectiveness. In *Proceedings of the 36th International Conference on Software Engineering*. 435–445.

[25] Kevin Jalbert and Jeremy S. Bradbury. 2012. Predicting mutation score using source code and test suite metrics. In *2012 1st International Workshop on Realizing AI Synergies in Software Engineering (RAISE '12)*. IEEE, 42–46.

[26] Yue Jia and Mark Harman. 2010. An analysis and survey of the development of mutation testing. *IEEE Transactions on Software Engineering* 37, 5 (2010), 649–678.

[27] 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*. 654–665.

[28] René Just, Franz Schweiggert, and Gregory M. Kapfhammer. 2011. MAJOR: An efficient and extensible tool for mutation analysis in a Java compiler. In *Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE '11)*. IEEE, 612–615.

[29] Pavneet Singh Kochhar, Ferdian Thung, and David Lo. 2015. Code coverage and test suite effectiveness: Empirical study with real bugs in large systems. In *Proceedings of the 2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER '15)*. IEEE, 560–564.

[30] Bob Kurtz, Paul Ammann, Marcio E. Delamaro, Jeff Offutt, and Lin Deng. 2014. Mutant subsumption graphs. In *Proceedings of the 2014 IEEE 7th International Conference on Software Testing, Verification, and Validation Workshops*. IEEE, 176–185.

[31] Bob Kurtz, Paul Ammann, and Jeff Offutt. 2015. Static analysis of mutant subsumption. In *Proceedings of the 2015 IEEE 8th International Conference on Software Testing, Verification, and Validation Workshops (ICSTW '15)*. IEEE, 1–10.

[32] Bob Kurtz, Paul Ammann, Jeff Offutt, Márcio E. Delamaro, Mariet Kurtz, and Nida Gökçe. 2016. Analyzing the validity of selective mutation with dominator mutants. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. 571–582.

[33] James R. Larus. 1999. Whole program paths. *ACM SIGPLAN Notices* 34, 5 (1999), 259–269.

[34] Aditya P. Mathur. 1991. Performance, effectiveness, and reliability issues in software testing. In *1991 Proceedings of the 15th Annual International Computer Software and Applications Conference*. IEEE Computer Society, 604–605.

[35] Akbar Siami Namin and James H. Andrews. 2009. The influence of size and coverage on test suite effectiveness. In *Proceedings of the 18th International Symposium on Software Testing and Analysis*. 57–68.

[36] A. Jefferson Offutt, Ammei Lee, Gregg Rothermel, Roland H. Untch, and Christian Zapf. 1996. An experimental determination of sufficient mutant operators. *ACM Transactions on Software Engineering and Methodology* 5, 2 (1996), 99–118.

[37] Mike Papadakis, Marinos Kintis, Jie Zhang, Yue Jia, Yves Le Traon, and Mark Harman. 2019. Mutation testing advances: An analysis and survey. In *Proceedings of the Advances in Computers*. Elsevier, 275–378.

[38] Mike Papadakis and Nicos Malevris. 2010. An empirical evaluation of the first and second order mutation testing strategies. In *Proceedings of the 2010 3rd International Conference on Software Testing, Verification, and Validation Workshops*. IEEE, 90–99.

[39] Mike Papadakis, Donghwan Shin, Shin Yoo, and Doo-Hwan Bae. 2018. Are mutation scores correlated with real fault detection? A large scale empirical study on the relationship between mutants and real faults. In *Proceedings of the 40th International Conference on Software Engineering*. 537–548.

[40] Domenico Serra, Giovanni Grano, Fabio Palomba, Filomena Ferrucci, Harald C. Gall, and Alberto Bacchelli. 2019. On the effectiveness of manual and automatic unit test generation: Ten years later. In *Proceedings of the 2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR '19)*. IEEE, 121–125.

[41] Donghwan Shin, Shin Yoo, and Doo-Hwan Bae. 2017. A theoretical and empirical study of diversity-aware mutation adequacy criterion. *IEEE Transactions on Software Engineering* 44, 10 (2017), 914–931.

[42] Akbar Siami Namin, James H. Andrews, and Duncan J. Murdoch. 2008. Sufficient mutation operators for measuring test effectiveness. In *Proceedings of the 30th International Conference on Software Engineering*. 351–360.

[43] Zhao Tian, Junjie Chen, Qihao Zhu, Junjie Yang, and Lingming Zhang. 2022. Learning to construct better mutation faults. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*. 1–13.

[44] Elaine J. Weyuker. 1993. Can we measure software testing effectiveness?. In *[1993] Proceedings of the 1st International Software Metrics Symposium*. IEEE, 100–107.

[45] W. Eric Wong and Aditya P. Mathur. 1995. Reducing the cost of mutation testing: An empirical study. *Journal of Systems and Software* 31, 3 (1995), 185–196.

[46] Martin R. Woodward and Michael A. Hennell. 2006. On the relationship between two control-flow coverage criteria: All JJ-paths and MCDC. *Information and Software Technology* 48, 7 (2006), 433–440.

[47] Peng Zhang, Yanhui Li, Wanwangying Ma, Yibiao Yang, Lin Chen, Hongmin Lu, Yuming Zhou, and Baowen Xu. 2020. Cbua: A probabilistic, predictive, and practical approach for evaluating test suite effectiveness. *IEEE Transactions on Software Engineering* 48, 3 (2020), 1067–1096.

[48] Peng Zhang, Yang Wang, Xutong Liu, Yanhui Li, Yibiao Yang, Ziyuan Wang, Xiaoyu Zhou, Lin Chen, and Yuming Zhou. 2022. Mutant reduction evaluation: What is there and what is missing? *ACM Transactions on Software Engineering and Methodology* 31, 4 (2022), 1–46.

Received 30 April 2023; revised 8 November 2023; accepted 22 November 2023