CBUA: A Probabilistic, Predictive, and Practical Approach for Evaluating Test Suite Effectiveness

Peng Zhang, Yanhui Li, Wanwangying Ma, Yibiao Yang, Lin Chen, Hongmin Lu, Yuming Zhou, and Baowen Xu, *Member, IEEE*

Abstract—Knowing the effectiveness of a test suite is essential for many activities such as assessing the test adequacy of code and guiding the generation of new test cases. Mutation testing is a commonly used defect injection technique for evaluating the effectiveness of a test suite. However, it is usually computationally expensive, as a large number of mutants (buggy versions) are needed to be generated from a production code under test and executed against the test suite. In order to reduce the expensive testing cost, recent studies proposed to use supervised models to predict the effectiveness of a test suite without executing the test suite against the mutants. Nonetheless, the training of such a supervised model requires labeled data, which still depends on the costly mutant execution. Furthermore, existing models are based on traditional supervised learning techniques, which assume that the training and testing data come from the same distribution. But, in practice, software systems are subject to considerable concept drifts, i.e., the same distribution assumption usually does not hold. This can lead to inaccurate predictions of a learned supervised model on the target code as time progresses. To tackle these problems, in this paper, we propose a Coverage-Based Unsupervised Approach (CBUA) for evaluating the effectiveness of a test suite. Given a production code under test, the corresponding mutants, and a test suite, CBUA first collects the coverage information of the mutated statements in the target production code under the execution of the test suite. Then, CBUA employs coverage to estimate the probability of each mutant being alive. As such, a mutation score is computed to evaluate the test suite effectiveness and the predicted labels (i.e., killed or alive) are obtained. The whole process only requires a one-time execution of the test suite against the target production code, without involving any mutant execution and any training data. CBUA can ensure the score monotonicity property (i.e., adding test cases to a test suite does not decrease its mutation score), which may be violated by a supervised approach. The experimental results show that CBUA is very competitive with the state-of-the-art supervised approaches in prediction accuracy. In particular, CBUA is shown to be more effective in finding mutants that are covered but not killed by a test suite, which is helpful in identifying the weaknesses in the current test suite and generating new test cases accordingly. Since CBUA is an easy-to-implement approach with a low cost, we suggest that it should be used as a baseline approach for comparison when any novel prediction approach is proposed in future studies.

Index Terms—Effectiveness, test suites, coverage, unsupervised model, mutation testing

1 Introduction

1.1 Motivation

In software testing, test suits are widely used to validate the correctness of software. Knowing the effectiveness (i.e., the fault detecting potential) of a test suite is essential for many activities such as assessing the test adequacy of code and guiding the generation of new test cases. Consequently, an immense amount of research effort so far has been devoted to developing an effective approach to evaluating the effectiveness of test suites [1], [2], [3], [4]. Mutation testing is widely considered as one of the most effective approaches t at finding faults [5], [6], [7], [8]. In mutation testing, a set of mutants (i.e., faulty variants) are generated by modifying a production code under test (CUT) in small ways (i.e., by applying mutation operators). A mutant is regarded as killed by a test suite if at least one test case in the test suite leads to the behavior of CUT to differ from the mutant. Otherwise, the mutant is regarded as survived (i.e., alive) in the test suite, which means that the test suite cannot detect bugs in the mutated source code. The ratio of killed mutants to all the non-equivalent mutants is referred as mutation score, which is used to assess test effectiveness. The higher the mutation score, the more effective the test suite is considered. In the literature [9], [46], mutation score is a commonly used indicator to evaluate the effectiveness of a given test suite.

Although mutation score is a good metric for measuring the effectiveness of a test suite in detecting defects, its computation can be expensive, especially for a large CUT [9], [46]. The reason is that a large number of mutants (buggy versions) are needed to be generated and executed against the test suite. In order to reduce the computation cost, recent studies used supervised models to predict the mutation score without executing the test suite against the mutants. Jalbert and Bradbury [10], [11] used a number of features extracted from both CUT and test code to build support vector machine (SVM) models to predict whether the resulting mutation score was "high", "medium", or "low". These features included static features (structural metrics collected from both production and test code) and dynamic features (code coverage metrics collected from production code when

• *The authors are with the State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, Jiangsu 210023, China. E-mail: {dz1833034, wwyma}@smail.nju.edu.cn,{yanhuili,yangyibiao,lchen,zhouyuming,bwxu}@nju.edu.cn, 32493172@qq.com.*

Manuscript received 7 Aug. 2019; revised 19 May 2020; accepted 13 July 2020. Date of publication 20 July 2020; date of current version 15 Mar. 2022. (Corresponding authors: Yanhui Li, Lin Chen, Yuming Zhou, and Baowen Xu) Recommended for acceptance by Raymond K. Sen. Digital Object Identifier no. 10.1109/TSE.2020.3010361

executing test cases against the CUT). They found that the proposed models were superior to a random guessing model. Grano *et al.* [12] used the features from more dimensions to build the supervised models, including production code smells, test code smells, and the readability of both production and test code. They reported that a supervised model built with only static features was good at predicting whether the resulting mutation score was "high" or "low". Unlike the direct prediction of mutation score, Zhang *et al.*'s PMT (Predictive Mutation Testing) approach [13] first used a supervised model built with the execution, infection, and propagation features to predict whether a mutant would be killed or remained alive. Then, a mutation score was computed based on the mutant-level prediction results. Compared with prior approaches, PMT has the following advantages. On the one hand, mutant-level prediction pinpoints the potential alive mutants, enabling developers to know the reason the current test suite is unable to detect defects. On the other hand, mutant-level prediction provides a quantitative mutation score, rather than a qualitative "high", "medium", or "low" tag. In this sense, PMT is more helpful for developers in test activities such as guiding the generation of new test cases and assessing the test adequacy of code.

Undoubtedly, predicting mutation score and mutant labels without executing mutants can reduce a large amount of computation cost for evaluating test suite quality and identifying killable mutants. In software testing activities, developers could benefit a lot from these studies, as it has been shown that supervised prediction models can achieve a promising prediction performance. However, the training of such a supervised model requires a sufficiently large amount of labeled data, which still depend on the costly mutant execution. In other words, existing studies focus on reducing the cost at the prediction phase but ignore the cost at the model building phase. Furthermore, such supervised models aim to apply the knowledge learned from historical data (either from the target project or external projects) to obtain a prediction for the target code. Since existing models are based on traditional supervised learning techniques, they assume that the training and the target data come from the same distribution [14]. But, in practice, "software systems are subject to considerable concept drifts", as "set of influencing features has changed" ("usually due to a change in the underlying bug generation process") [15]. Consequently, the same distribution assumption rarely holds. This can lead to inaccurate predictions of a learned supervised model as time progresses.

To tackle the above-mentioned problems, in this paper, we propose a Coverage-Based Unsupervised Approach (CBUA) for evaluating the effectiveness of a test suite. Given a target production code under test, the corresponding mutants, and a test suite, CBUA first collects the coverage information of the mutated statements in the target production code under the execution of the test suite. Then, CBUA employs the coverage information to estimate the survival probability of each mutant. Based on this information, on the one hand, the expected number of alive mutants can be computed and hence a mutation score can be obtained. On the other hand, the label (i.e., killed or alive) of each mutant can be obtained by comparing its survival probability against the threshold 0.5. The whole process only requires a one-time execution of the test suite against the target production code under test, without involving any mutant execution and any training data. As a result, CBUA has the following main advantages over existing supervised prediction models: first, it does not need any training data; second, it has a low model building cost; third, it is not subject to the concept drift challenge. In order to evaluate the effectiveness of CBUA, we use a large number of real world projects to conduct the experiment. The experimental results show that CBUA is very competitive to the state-of-the-art supervised approaches in terms of the prediction accuracy. In particular, CBUA is superior in finding mutants that are covered but not killed by test suites at hand, which is helpful in guiding developers to generate new test cases. Since CBUA is an easy-to-implement model with a low cost, we suggest that it should be used as a baseline model for comparison when any novel prediction approach is proposed in the future studies. This will help advance the progress in predictive mutant testing.

1.2 Contribution

In this study, we make the following main contributions:

- We propose an *unsupervised approach* CBUA to predict the mutation score of a test suite *without involving any mutant execution*. CBUA is only based on the coverage information of the mutated statements in the target production code under the execution of a test suite. Different from existing supervised approaches, CBUA does not need any training data and any mutant execution. As a result, it has a low cost to apply and is free of the challenge on different data distributions of the training and target test data usually caused by considerable concept drifts in software systems.
- On the methodological front, we propose a probabilistic formula to estimate the survival probability of each mutant based on coverage information. On the one hand, this enables CBUA to obtain a predicted mutation score by computing the excepted number of alive mutants. On the other hand, this enables CBUA to predict whether a mutant will be killed or remained alive by comparing its survival probability against the probability threshold 0.5. Consequently, similar to the state-of-the-art supervised approach PMT, CBUA also has the ability to pinpoint the potential alive mutants, thus helping "developers to locate the weaknesses in the current test suite and to add new test cases accordingly" [13]. In particular, we theoretically show that CBUA can ensure the score monotonicity property (i.e., adding test cases to a test suite does not decrease its mutation score), which may be violated by a supervised approach.
- We conduct an extensive study to investigate the usefulness of CBUA in practice. In order to obtain reliable conclusions, we use two large-scale data sets, one from 163 open-source projects [13] and another one from 654 open-source projects [44], to conduct the experiments. Compared with the-state-of-the-art supervised approach PMT, the experimental results consistently show that: (1) CBUA is very competitive in predicting mutation score; (2) CBUA has a similar

ability of pinpointing alive mutants but exhibits a more trustworthy predictability (i.e., how confident a classifier is in classifying a mutant); and (3) CBUA is more effective in finding mutants that are covered but not killed by a test suite. Taken together, these evidences demonstrate the usefulness of CBUA.

1.3 Organization

The rest of this paper is organized as follows. Section 2 introduces the background. Section 3 describes our proposed CBUA approach. Section 4 presents the data sets and the experimental design. Section 5 reports the experimental results. Section 6 discusses the experimental results. Section 7 analyzes the threats to the validity of our study. Section 8 concludes the paper and outlines directions for future work.

2 Background

In this section, we first introduce the mutation testing context. Then, we overview the work related to effectiveness evaluation of test suites. Finally, we compare the advantages and disadvantages of supervised models with unsupervised models for predicting the test suite effectiveness.

2.1 Mutation Testing

Mutation testing is a testing method that mutates (changes) statements in CUT to obtain faulty versions (i.e., mutants) and checks if a test suite is able to find the faults in mutants. It is important to point out that, the mutation testing techniques discussed in this paper are all monomorphic, i.e., only one modification is made at a statement each time when a mutant is generated. As a result, each mutant corresponds to a mutated statement in the CUT. A mutant is generated by a mutation operator, in compliance with a transformation rule such as replacing the operator '-' with '+'. In this case, the test result may differ between a mutant and the CUT if a statement includes '-' has been executed. If the difference exists, we say that the test has killed the mutant. If all the tests fail to kill the mutant, we say that the mutant survives. There are two contexts of mutation testing: strong mutation testing and weak mutation testing. In the first context, killing means the mutant and the original program generate different outputs. While in the second context, killing means the program state of the mutant program differs from the original one at some point during execution. It can be safely concluded that, in weak mutation testing, the two outputs can be the same. This paper focuses on the discussion of strong mutation testing. A common indicator for evaluating the effect of a mutation test is the mutation score, defined as the percentage of killed mutants among the total number of non-equivalent mutants (a mutant is called "equivalent" if its output cannot be distinguished from the output of the original program by test cases). A higher mutation score indicates a more effective test [7], [16], [17], [18], [19], [20].

2.2 Test Suite Effectiveness Evaluation

Test suite effectiveness evaluation is a fundamental task in software testing, which is "central to development of automated test-generation techniques" for generating high-quality suites [21]. Ideally, test suite effectiveness can be measured by the number of real faults uncovered by a test suite. However, before testing, real faults are unknown in advance. Consequently, it is hard, if not possible, to apply the number of uncovered real faults as an effectiveness metric to assess and improve the effectiveness of the current test suite.

Code coverage is a widely used metric for evaluating the effectiveness of a test suite. The rationale behind this is that faults cannot be detected if the code is not covered by a test suite. Intuitively, a test suite with a higher code coverage may have a higher probability to uncover faults. In the past years, many studies have been conducted to empirically investigate the influence of code coverage on test effectiveness. Overall, it has been shown that there is a correlation between code coverage and test effectiveness, even if the test suite size is controlled as a confounding factor [22], [23], [24], [25], [26]. In many cases, they exhibit a non-linear correlation with a varying strength. Nonetheless, it has been pointed out that code coverage is not enough for evaluating test suite effectiveness, as "it fails to check whether the program executions generated by the test suite were actually correct" [27]. That is to say, "an execution that produces a wrong output that is unnoticed by the test suite is counted exactly the same as an execution that produces the right output for coverage" [27].

Mutation score provides an alternative solution to evaluate the effectiveness of a test suite. In nature, mutation score quantifies the ability of a test suite to detect artificial faults (i.e., mutants). As stated in [27], "the assumption is that the more effective a test is in finding artificial faults, the more effective it would be in finding real faults". Recently, several studies empirically showed that mutant detection was significantly related to real fault detection [28], [29]. In particular, mutation score had a stronger correlation with real fault detection than the correlation between code coverage and real fault detection [29]. This means that mutation score can be used a proxy to reflect the ability to detect real faults. Compared with code coverage testing, the advantage of mutation testing is that it can tell what part of the code is not well tested, rather than what part of code is not being tested [30].

However, the cost of computing mutation score is far greater than that of computing coverage, as the test suite needs to be executed against a large number of mutants. As a result, whether we can predict the mutation score by existing data instead of executing all mutants has attracted more and more attention. Currently, a few studies have been devoted to developing supervised models to predict mutation score without mutant execution [10], [11], [12], [13]. At a high level, these studies take a two-phase framework. At the first phase, a model is trained on the training data with a supervised modeling technique. At the second phase, the trained model is applied to predict mutation score.

- CUT-level prediction. Jalbert and Bradbury predicted whether a CUT (i.e., a class or method) had a "high", "medium", or "low" mutation score [10], [11]. The used features include source code metrics (from CUT), test suite metrics (from test code), and coverage metrics (from the execution of the test suites against CUT). To the best of our knowledge, this is the first attempt to explore whether mutation score can be predicted. They achieved an accuracy

around 55 percent, superior to a random guessing model (33 percent). Their work showed the feasibility of predicting mutation scores and left much room for improvement in effectiveness.

- Test-case-level prediction. Grano *et al.* [12] did a similar work to Jalbert and Bradbury's work [10], [11]. The main differences are three-fold. First, Grano *et al.* used more feature dimensions to build the supervised models, including code smells (from CUT), test smells (from test code), and readability (from CUT and test code). Second, they assigned test cases to one of the two sets using the quartiles of the mutation score: "those falling within the first quartile are assigned to the non-effective set, while the ones in the fourth quartile to the effective set" [12]. In other words, they treated mutation score prediction as a two-classification problem. Third, they predicted mutation score for each test case, rather than for CUT. Based on cross-validation, they achieved performance close to 0.95 in term of F_1 and AUC for a model relying on both dynamic and static features, while above 0.80 only using static features. It is noteworthy that such good results were based on selectively discarding half of the test cases. In other words, before predicting, they discarded test cases "having an average effectiveness" (those falling between the first and fourth quartiles) and focused "only on those that can be considered as having a high-or low-quality". In practice, however, it is hard, if not impossible, to identify and exclude those test cases in advance. This means that their results may be optimistic and need to be further investigated in the future.
- Mutant-level prediction. Unlike the direct prediction of mutation score, Zhang *et al.*'s PMT (Predictive Mutation Testing, the default modeling technique is random forest) approach [13] first used the execution, infection, and propagation features to predict whether a mutant would be killed or remained alive. Then, the ratio of predicted killed mutants to all mutants was computed as their predicted mutation score. Compared with Jalbert and Bradbury's and Grano *et al.*'s approaches, PMT not only can enable developers to pinpoint alive mutants but also can provide a quantitative mutation score. In this sense, PMT is more helpful for developers in test activities such as guiding the generation of new test cases. PMT was able to get good effectiveness with F_1 and AUC values above 0.85 for predicting mutant labels, with less than 0.10 absolute prediction error in mutation score. More recently, Mao *et al.* conducted an extensive study to investigate the effectiveness of PMT by including more features and considering powerful deep learning techniques (abbreviated as "Ext-PMT" in the following) [44]. Based on a 5-fold cross-validation on 654 projects, they showed that random forest was one of the best modeling techniques for Ext-PMT, leading to an average F_1 of 0.69, an average AUC of 0.78, and an average absolute prediction error of 0.15 in mutation score.

The existing approaches are all supervised approaches, which require a lot of labeled data to train prediction models. In real software development, software systems are subject to considerable concept drifts. Consequently, it is not possible that a model is trained only once and used forever. In other words, there is a need to use new training data to update the model frequently. In this context, a huge cost will be spent on collecting the training data, as the label information is based on mutant execution. Furthermore, as we will show in Section 6, a supervised approach may violate the score monotonicity property (i.e., adding test cases to a test suite does not decrease its mutation score). In contrast, our proposed CBUA is an unsupervised approach without any mutant execution. In particular, CBUA satisfies the score monotonicity property (see Section 3 for the proof). Like PMT, CBUA is at the mutant level, which can be easily used to predict a mutation score for CUT, each single test case, or a test suite.

2.3 Supervised versus Unsupervised Approaches

Supervised learning aims to learn a function that maps an input to an output based on example input-output pairs [31]. The common supervised techniques include logistic regression, random forest (RF), and support vector machine (SVM). To build a supervised model, a large amount of labeled data is often required, which may incur a significant data collection cost. In contrast, unsupervised learning aims to draw inferences from unlabeled input datasets. The common unsupervised techniques include clustering and association rule mining.

Since prior knowledge from the training data is used, it is often expected that a supervised model has a higher prediction performance than an unsupervised model. However, an inherent challenge for supervised learning is that the training data and the testing data may exhibit different distributions. As a result, the prior knowledge from the training data may not be well applied to the testing data. Indeed, it is not uncommon to see such a phenomenon, as software systems are subject to considerable concept drifts [15]. In this case, an unsupervised model is a good alternative for practitioners if it can achieve a prediction performance comparable to a supervised model. The reason is that there is no need to collect the labeled training data, thus saving a large data collection cost.

In our prior work, we found that, in defect prediction field, simple unsupervised models on the testing data had a prediction performance comparable or even superior to most of the existing supervised models in the literature [32]. Inspired by this work, we want to explore whether the same phenomena can be observed in test suite effectiveness evaluation. If we can find an unsupervised approach without any mutant execution that has a comparable prediction performance to the-state-of-the-art supervised approach, this will lead to a large benefit for practitioners in mutation testing.

3 CBUA: Coverage-Based Unsupervised Approach

For practitioners, a mutant-level prediction is more informational for identifying weak test cases and for guiding the generation of new powerful test cases. Therefore, we decide to design CBUA as a mutant-level prediction. The core idea is to use the probability theory to develop a formula to estimate the survival probability of each mutant based on the test coverage of the mutated statements in CUT under the execution

of a test suite. The intuition behind this is that, the more test cases have covered a mutated statement, the more likely to be killed the corresponding mutant. After obtaining the survival probability of each mutant, we can compute the expected number of alive mutants in CUT under the test suite. As a result, a mutation score can be easily obtained, indicating the effectiveness of the test suite under study. Furthermore, by comparing the survival probability of a mutant against the threshold 0.5, we can obtain its label, i.e., whether the mutant will be killed or remain alive. The whole process only requires a one-time execution of the test suite against CUT, without involving training data and mutant execution.

In the next, we describe the framework of CBUA as well as the frameworks of a traditional approach and a supervised approach for evaluating the effectiveness of a test suite. For the traditional approach, at the first step, execute the test suite on CUT as well as on the corresponding mutants. At the second step, compare their outputs to obtain the labels of mutants. At the third step, compute a mutation score based on the mutant labels. For a supervised approach, at the first step, use labeled historical data (i.e., mutant-level metrics and labels) to train a supervised model. At the second step, collect dynamic features by executing the test suite on CUT or static features by statically analyzing the test suite and mutants. At the third step, use the above features as the input of the trained model to conduct a prediction, including a mutation score and mutant labels. On the one hand, compared with the traditional approach, CBUA eliminates the cost of running the test suites against the mutants. On the other hand, compared with the supervised approach, CBUA eliminates the cost of collecting labeled data for training a model. Specifically, CBUA consists of the following key steps:

- Step 1: execute the test suite on CUT to collect the coverage information of the mutated statements corresponding to the mutants.
- Step 2: use the resulting coverage information to estimate the survival probability of each mutant.
- Step 3: two actions will be taken at this step. On the one hand, compute the mutation score (based on the number of alive mutants estimated using the survival probability of each mutant). On the other hand, classify mutants into either "alive" or "killed" (a mutant with a survival probability larger than 0.5 will be labeled as "alive" and "killed" otherwise) and report their predictability (i.e., the confidence in classifying a mutant).

In the following, we first describe in detail the key steps in CBUA. Then, we prove that CBUA satisfies the score monotonicity property (i.e., adding test cases to a test suite does not decrease its mutation score), which is a property any mutation score estimation approach should have. Finally, we analyze the relation of CBUA with the simple code coverage approach. To see the comparison figure of CBUA with traditional and supervised approaches in appendix, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TSE.2020.3010361.

3.1 Coverage Metric Collection

We use only one metric to measure the coverage strength for each mutated statement in CUT when executing the test suite against CUT: *numTestCovered*, the number of test cases covering the mutated statement (it was called an execution feature in [13]) in CUT. For the simplicity of presentation, in the following, we use the word "a mutated statement covered" and "a mutant covered" interchangeably, as there is a corresponding relationship between the mutated statement and the associated mutant.

Intuitively, on the one hand, if a mutated statement in CUT is not covered by the test suite, CUT and the corresponding mutant will have the same output. Consequently, the corresponding mutant is impossible to be killed by the test suite [33], [34], [35]. On the other hand, if a mutated statement in CUT is covered by the test suite, the corresponding mutant is likely to be killed. In particular, mutants covered by more test cases are more likely to be killed. This is the internal logic of our choice of this metric to measure the coverage strength. In practice, it is easy to use (or extend) existing code testing tools to collect *numTestCovered*.

3.2 Survival Probability Estimation

For the simplicity of presentation, assume that there are a total of N mutants, of which n_1 are actually killed and n_2 are actually survived under the test suite. As a result, we have $N = n_1 + n_2$. Furthermore, assume that X is the number of mutants whose corresponding mutated statements are not covered by any test case. According to this assumption, all the X mutants must be alive under the test suite. Consequently, we can conclude that $X \leq n_2$.

For a given mutation m, we attempt to use the associated *numTestCovered* to estimate its survival probability *prob*. In our study, we consider mutation testing as a Bernoulli experiment. In other words, we assume that every time a mutated statement is covered by a test case, the corresponding mutant survives with a probability of p. Consequently, if a mutant m is covered by x test cases (i.e., its *numTestCovered* is equal to x), then it should have a survival probability:

$$prob = p^x. \tag{1}$$

However, many studies report that there is a significant diminishing marginal utility in test suite [53]. Specifically, the first few test cases can kill a large number of mutants, while the later test cases can kill fewer new mutants. This phenomenon shows that the utility of test cases is decreasing. To model this phenomenon, we use a utility function to indicate the "real size" of test cases (instead of using x directly), which satisfies the following two properties: 1) monotonically increasing; and 2) second derivative < 0. Furthermore, many recent studies show that test effectiveness is not linearly related to the number of test cases [22], [23], [36]. In particular, Inozemtseva and Holmes [22] showed that there was a logarithmic relationship between them in most cases. Due to the above reasons, for a given mutant m covered by x test cases, we redefine its survival probability as:

$$prob = p^{log_2(x+1)}. \tag{2}$$

In equation (2), $log_2(x+1)$ is a utility function used to characterize the diminishing marginal utility phenomenon of test cases in uncovering defects.

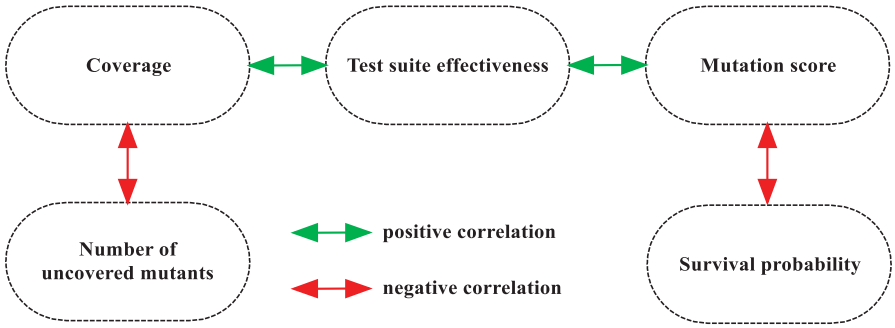


Fig. 1. Correlation among several measures.

As can be seen, in order to compute *prob*, we first need to obtain *p*. As shown in Fig. 1, our intuition and existing empirical evidence in the literature show that: (1) mutant coverage is negatively related with the number of uncovered mutants; (2) mutation score is negatively related with survival probability; and (3) both mutant coverage and mutation score are positively related with test suite effectiveness. Consequently, survival probability is expected to be positively related with the number of uncovered mutants. Considering that the two metrics' ranges are the same, we hence directly use *X*/*N* to approximate *p* in our study. As a result, we have:

$$p = \frac{X}{N}. \tag{3}$$

3.3 Mutation Score Prediction

Using *numTestCovered*, we can divide the N mutants into k equivalent groups (i.e., within each group, all the mutants have the same *numTestCovered* value). Let c_j be the number of mutants and x_j be the *numTestCovered* of a mutant in the jth group, $1 \leq j \leq k$. Furthermore, we assume that $x_1 < , \ldots, < x_j < , \ldots, < x_k$. For each of the c_j mutants in the jth group, since the corresponding mutated statement is covered by x_j test cases, it has a survival probability of $p^{log_2(x_j+1)}$. Given the c_j mutants, the number of surviving mutants should be $c_j p^{log_2(x_j+1)}$. Consequently, for a project, the number of alive mutants E_{alive} under the test suite can be estimated via the following formula:

$$E_{alive} = \sum_{j=1}^{k} c_j p^{log_2(x_j+1)}. \tag{4}$$

As such, the predicted mutation score *pms* can be computed as follows:

$$pms = \frac{N - E_{alive}}{N}. \tag{5}$$

We next use an example to elaborate the computation process of the predicted mutation score. The upper part of Fig. 2 shows for each equivalent group (j) on the project uaa (taken from the shared data online by [13]) the corresponding *numTestCovered* (x_j), survival probability (prob), and the number of mutants (c_j). Note that the noise data have been filtered out by the method described in Section 4.4. As can be seen, $N = 5975$ and $X = 2842$. Therefore, we have:

$$p = \frac{X}{N} = 0.476.$$

According to equation (4), the estimated number of alive mutants should be:

$$E_{alive} = \sum_{j=1}^{k} c_j \ p^{log_2(x_j+1)} = 3479.$$

As such, according to equation (5), we have:

$$pms = \frac{N - E_{alive}}{N} = \frac{5975 - 3479}{5975} = 0.4177.$$

By checking the real label, we find that, of the 5975 mutants, 2168 mutants are actually "killed" (i.e., $n_1 = 2168$), while 3807 mutants are actually "alive" (i.e., $n_2 = 3807$). Therefore, the actual mutation score *ams* is:

$$ams = \frac{2168}{5975} = 0.3628.$$

Consequently, the absolute prediction error APE is:

$$\begin{aligned} \text{APE} &= |pms - ams| \\ &= |0.4177 - 0.3628| \\ &= 0.0549. \end{aligned}$$

3.4 Mutant Label Prediction

For a given mutant, its label prediction is straight: if the estimated survival probability is larger than 0.5, it will be classified as "alive"; otherwise, it will be classified as "killed". Furthermore, a predictability value is provided to help understand the confidence that CBUA is in classifying the mutant. Similar to [13], we use the distance between the estimated survival probability and 0.5 to quantify predictability:

$$predictability = |prob - 0.5|. \tag{6}$$

Since *prob* has a range between 0 and 1, predictability will have a range between 0 and 0.5. A higher predictability indicates a higher confidence in classifying a mutant. As stated in [13], "with such information, developers may learn the degree of certainty of the prediction on each mutant and choose to either believe in the predictive result when the predictability is high, or to doubt the predictive result and

Group(j)	1	2	3	4	5	6	...	47
numTestCovered(x_j)	0	1	2	3	4	5	...	75
Survival probability(prob)	1	0.476	0.226	0.108	0.051	0.024		0
Number of mutants (c_j)	2842	665	300	264	315	244	...	3
Cumulation of mutants	2842	3507	3807	4071	4386	4630	...	5975

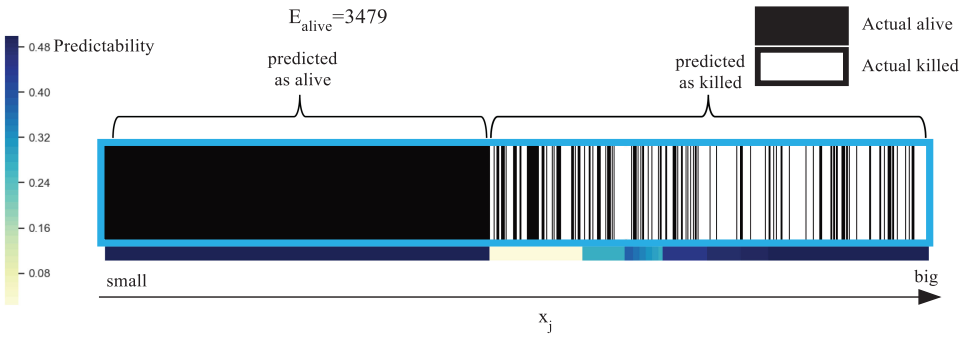


Fig. 2. An example project *uaa* illustrating the prediction process of test suite effectiveness in CBUA.

run the mutant against the tests to get more accurate results when the predictability is low."

In summary, each mutant not only will have a predicted label but also will have a predictability value.

We next use the project *uaa* shown in Fig. 2 to illustrate mutant label prediction. We can see that, the lower part of Fig. 2 reports the results of mutant label prediction, including the actual and predicted labels. In particular, the narrow colorful belt below the blue box visualizes the corresponding predictability. Here, we make two important observations. On the one hand, all the uncovered mutants have the same "alive" label and the same predictability of 0.5 (the highest prediction confidence). On the other hand, those mutants with the same predicted label (for example, the same "killed" label) may exhibit different predictability, providing developers valuable information to make further test decisions.

3.5 Score Monotonicity Property Assurance

Score monotonicity property denotes that adding test cases to a test suite does not decrease its mutation score. Intuitively, a reasonable mutation score prediction approach should have the score monotonicity property. Otherwise, it may lead to counter-intuitive prediction conclusions. Indeed, increasing the size of a test suite is an important way to improve test effectiveness. If CBUA has the score monotonicity property, we can know that it is at least more reasonable than those prediction approaches without such a property.

We next prove that CBUA satisfies the score monotonicity property. Assume that pms' is the mutation score after adding test cases to the test suite, i.e.,

$$pms' = \frac{N - E'_{alive}}{N}$$

As can be seen, in order to prove that the score monotonicity property holds, we just need to prove that $pms' \geq pms$. In other words, we only need to prove that $E_{alive} \geq E'_{alive}$.

Recall that X is the number of uncovered mutants, p is the survival probability of a mutant when it is covered by a test case, c_j is the number of mutants in the jth group, and x_j is the number of test cases that a mutant in the jth group is covered. We assume that, after adding test cases, the corresponding number of uncovered mutants is X' and the corresponding survival probability of a mutant is p'.

Since adding test cases does not increase the number of uncovered mutants, we have:

$$X \geq X\prime.$$

Then, we have:

$$p = X/N \geq X\prime/N = p\prime.$$

After adding test cases, the jth original equivalent group will be broken into i new equivalent groups. Assume that c'_{jm} is the number of mutants in the mth new group and x'_{jm} is the number of test cases that a mutant in the mth new group is covered, $1 \leq m \leq i$. Then, we have:

$$c_j = \sum_{m=1}^{i} c'_{jm}.$$

For any mutant, adding test cases does not decrease the number of test cases covering the mutant. Therefore, we have:

$$x'_{jm} \geq x_j, \text{for} 1 \leq m \leq i.$$

Consequently,

$$p^{log_2(x_j+1)} \quad p^{log_2(x'_{jm}+1)}, \text{for} 1 \leq m \leq i.$$

As a result, we have:

$$\begin{aligned} c_j \quad p^{log_2(x_j+1)} &= \left[\sum_{m=1}^{i} c'_{jm}\right] p^{log_2(x_j+1)} \\ &= \sum_{m=1}^{i}\left[c'_{jm}\, p^{log_2(x_j+1)}\right] \\ &\geq \sum_{m=1}^{i}\left[c'_{jm}\, p^{log_2\left(x'_{jm}+1\right)}\right] \\ &\geq \sum_{m=1}^{i}\left[c'_{jm}\, (p')^{log_2\left(x'_{jm}+1\right)}\right]. \end{aligned} \tag{7}$$

When summing the above inequation over all the k original equivalent groups ($1 \leq j \leq k$), we can find that: (1) the resulting left item is E_{alive}; and (2) the resulting right item is E'_{alive}. Therefore, we have:

$$E_{alive} \geq E\prime_{alive}.$$

This means that adding test cases to a test suite does not increase the estimated number of alive mutants and hence does not decrease the predicted mutation score. In other words, for CBUA, the score monotonicity property holds.

3.6 Relationship With Simple Code Coverage Approach

Given the coverage information of the mutated statements in CUT under a test suite, a simple unsupervised approach to classifying the mutants is as follows: all the uncovered mutants are classified as "alive", while all the covered mutants are classified as "killed". Based on the predicted "killed" mutants, a mutation score can be computed (i.e., number of predicted "killed" mutants / the total number of mutants). For the simplicity of presentation, we name this simple code coverage approach as "COV". In [13], Zhang *et al*. used COV as a baseline approach to investigate the effectiveness of PMT.

From the viewpoint of mutant classification, CBUA can indeed be regarded as an improved version of COV. In CBUA, a mutant is classified as "alive" if the corresponding survival probability $prob = p^{log_2(x+1)}$ is larger than 0.5 and is classified as "killed" otherwise. Recall that $p = X/N$ and x is the number of test cases covering the mutant. In the following, we analyze the relationship between CBUA and COV under the following two cases:

(1) case 1: for a project with $p < 0.5$

In CBUA, all the uncovered mutants (i.e., $x = 0$) must be classified as "alive", as they have a survival probability:

$$prob = p^{log_2(x+1)} = p^{log_2(0+1)} = 1.$$

All the covered mutants (i.e., $x \geq 1$) must be classified as "killed", as they have a survival probability:

$$prob = p^{log_2(x+1)} \quad p^{log_2(1+1)} = p < 0.5.$$

As can be seen, in this case, CBUA produces the same classification as COV on this project.

(2) case 2: for a project with $p \geq 0.5$

In CBUA, all the uncovered mutants must be classified as "alive" (same as done in COV). For a covered mutant, whether it will be classified as "alive" or "killed" depends on the value of p and the value of x. If p is large enough and x is small enough, the mutant will have a survival probability larger than 0.5 and hence will be classified as "alive". Otherwise, the mutant will be classified as "killed". In COV, all the covered mutants are simply classified as "killed". Intuitively, it is possible that many covered mutants have a real label of "alive". In other words, COV may lead to misclassification: actually "alive" mutants misclassified as "killed". However, CBUA has a potential to alleviate such a misclassification.

By the above analysis, we can see that: (1) CBUA produces the same classification as COV on a project with $p < 0.5$; (2) CBUA has a potential to alleviate the misclassification of COV on a project with $p \geq 0.5$. Note that, in both cases, CBUA can provide developers a predictability value for each prediction, informing the prediction confidence. However, COV is unable to provide such a predictability value.

From the viewpoint of mutation score computation, CBUA takes a different computation method compared with COV. CBUA first uses the survival probability of each mutant to estimate the number of alive mutants and then uses this information to compute a mutation score. As can be seen, this computation process does not involve mutant classification. However, COV first classifies mutants into "alive" or "killed" and then uses the number of predicted "killed" mutants (i.e., $N - X$) to compute a mutation score. Numerically, COV uses the following formula to compute a mutation score:

$$pms(COV) = \frac{N - X}{N}.$$

CBUA uses the following formula to compute a mutation score:

$$pms(CBUA) = \frac{N - E_{alive}}{N} = \frac{N - \sum_{j=1}^{k} c_j \times p^{log_2(x_j+1)}}{N}.$$

Since $c_1 = X$ and $x_1 = 0$, we have:

$$pms(CBUA) = \frac{N - \left[X + \sum_{j=2}^{k} c_j \times p^{log_2(x_j+1)}\right]}{N}.$$

Consequently, we have $pms(CBUA) \leq pms(COV)$. This means that COV produces a larger predicted mutation score than CBUA. The reason is that, in COV, only all the X uncovered mutants are classified as "alive", which ignores the fact that many covered mutants are also actually "alive". In this sense, CBUA has a potential to produce a more accurate mutation score than COV.

Consider the project uaa shown in Fig. 2. Since $N = 5975$ and $X = 2842$, $p = X/N = 0.476$. Consequently, CBUA produces the same classification as COV on this project. Furthermore, we have:

$$pms(COV) = 0.5244.$$

As shown in Section 3.3, we have:

$$pms(CBUA) = 0.4177 ams = 0.3628.$$

Therefore, COV leads to an absolute prediction error $|0.5244 - 0.3628| = 0.1616$, while CBUA leads to an absolute prediction error $|0.4177 - 0.3628| = 0.0549$. As can be seen, CBUA is superior to COV in mutation score prediction.

Zhang *et al*. [13] showed that PMT was superior to COV in mutation score prediction. Given this result, an interesting problem is immediately raised: what if we compare CBUA with PMT in mutation score prediction? If CBUA can achieve a competitive performance, then it will provide an easier-to-use approach to evaluating test suite effectiveness. We will empirically investigate this problem in the next sections.

TABLE 1
The Features Used in O-PMT

	Metric	Definition
Execution features	numExecuteCovered	How many times the mutated statement is executed by the whole test suite
	numTestCovered	How many tests from the test suite cover the mutated statement
Infection features	typeSatement	The type of the mutated statement
	typeOperator	The type of the mutation operator
Propagation features	infoComplexity	The McCabe Complexity of the mutated method
	deplnheritance	The maximum length of a path from the mutated class to a root class in the inheritance structure.
	depNestblock	The depth of nested blocks of the mutated method.
	numChildren	The total number of direct subclasses of the mutated class.
	LOC	The number of lines of code in the mutated method.
	Ca	The number of classes outside the mutated package that depend on classes inside the package.
	Ce	The number of classes inside the mutated package that depend on classes outside the package.
	instability	Which is an indicator of the packag's resilience to change, and is computed based on the Ca and Ce values, i.e.,, $Ce/(Ce + Ca)$.
	typeRetum	The return type of the mutated method
	numMutantAssertion	The total number of assertions in the test methods that cover each mutant.
	numClassAssertion	The total number of test assertions inside the mutated class's corresponding test class.

4 Experimental Design

In this section, we describe in detail the experimental design. First, we introduce the baseline approaches that CBUA will be compared against. Next, we list the research questions under investigation. Then, we characterize the data sets and describe the data preprocessing approach used in our study. After that, we report the performance evaluation method that we use. Finally, we present the data analysis method.

4.1 Baseline Approaches

We use COV and PMT [13], two mutant-level prediction approaches, as the baseline approaches to investigate the effectiveness of CBUA. Given a CUT, the corresponding mutants, and a test suite, COV and PMT take the following steps to evaluate test suite effectiveness.

- Baseline 1: COV. First, execute the test suite on CUT. Then, a mutant is classified as "killed" if the corresponding mutated statement in CUT is covered by the test suite and is classified as "alive" otherwise. Finally, use the predicted labels of mutants to evaluate the effectiveness of the test suite.
- Baseline 2: PMT. Our study uses two versions of PMT: the original PMT (O-PMT for short) [13] and a variant called Ext-PMT [44]. They use the same framework to build the model: first, train a random forest model on the training data (the default modeling technique is random forest); then, apply the model to classify whether a mutant is "killed" or "alive". Unlike CBUA, for PMT built with random forest, it is very hard, if it is not impossible, to explicitly present the relationships between inputs and the output. Finally, use the predicted labels of mutants to evaluate the effectiveness of the test suite. Their difference is that: the original PMT employs 15 features (shown in Table 1) related to execution, infection, and propagation to build the model [13], while Ext-PMT uses 95 features (by adding a large number of static structural features) to build the model. In the following, when needed, we will explicitly state whether PMT denotes O-PMT or Ext-PMT. In Table 1, the first column shows the category of features. The second and third columns respectively show the name and definition of the features.

The reasons for choosing COV and PMT as the baselines are two-fold. On the one hand, COV is an easy-to-implement unsupervised approach. If CBUA is not superior to COV, it appears that CBUA is of little value in practice. On the other hand, PMT is a newly proposed supervised approach. By comparing CBUA against PMT, we can know how well CBUA performs compared with the state-of-the-art approaches. If the results show that CBUA is competitive, it would be better to apply the unsupervised CBUA rather than the supervised PMT in practice when taking into account the application cost (including metrics collection cost and model building cost). In total, choosing COV and PMT as the baseline approaches helps determine the actual usefulness of CBUA.

4.2 Research Questions

We attempt to investigate the following research questions to understand the actual usefulness of CBUA.

RQ1 (Effectiveness of CBUA): How well does CBUA perform in test suite effectiveness prediction (as well as in mutant label prediction) compared with the baseline approaches?

The purpose of RQ1 is to investigate how well CBUA performs compared with COV and PMT. On the one hand, based on the analysis in Section 3.6, it is expected that CBUA is superior to COV. On the other hand, since PMT is a supervised approach, it is expected that PMT is superior

to the unsupervised approach CBUA. However, if PMT is only marginally better than or is similar to CBUA, it would be a good option for practitioners to apply CBUA in practice. We use the data sets shared in Zhang *et al.*'s study [13] and Mao *et al.*'s study [44] to investigate RQ1 (mutants generated by *PIT* and test suites written by manual). The answer to RQ1 enables us to understand whether it is necessary to use the new approach CBUA in practice.

RQ2 (Influence of application factors): How do different mutation testing tools, test types, and mutation magnitude influence the prediction performance of CBUA?

The purpose of RQ2 is to investigate the generalizability of CBUA. Once established the value of CBUA in test suite effectiveness prediction relative to the baseline approaches COV and PMT, it is natural to investigate how our finding is influenced by important application factors including mutation tools, test types, and mutation magnitude. The answer to RQ2 enables us to know whether CBUA is widely applicable when we use different mutation testing tools to generate mutants, use different ways to generate test suites, or use high-order mutant tools to generate mutants.

RQ3 (Mutant predictability and its trustworthiness): What is the predictability of the mutants and how trustworthy is it when using CBUA to predict their labels?

The purpose of RQ3 is to understand the distribution and trustworthiness of mutant predictability. When predicting the label of a mutant, a high predictability indicates a high confidence that CBUA is in classifying the mutant. However, it is possible that a mutant has a high predictability but the predicted label and the actual label are different. As a result, there is a strong need to evaluate the trustworthiness of mutant predictability, i.e., the validity that a high predictability indicates a more accurate prediction. In Zhang *et al.*'s study [13], the distribution of mutant predictability under PMT was analyzed but its trustworthiness was not investigated. We want to know whether CBUA has a higher trustworthy predictability than PMT. Therefore, we use PMT as the baseline approach to investigate the trustworthiness of mutant predictability under CBUA. The answer to RQ3 enables us to know whether CBUA can provide useful predictability information for developers to make subsequent testing decisions.

RQ4 (Practical benefits): How well does CBUA guide in finding mutants that are covered but not killed by a test suite?

The purpose of RQ4 is to investigate the practical benefits of CBUA. Knowing those mutants covered but not killed by a test suite (abbreviated as "covered but alive mutants") is helpful for developers to understand the weaknesses in the current test suite and design new test cases accordingly. To this end, we should filter out all the uncovered mutants for analysis. The reason is that uncovered mutants are definitely alive mutants. Given all the covered mutants, it is natural to investigate which approach (COV, CBUA, or PMT) is more effective in guiding the discovering of "covered but alive mutants". Since COV predicts all the covered mutants as "killed" mutants, it is unable to help developers find "covered but alive mutants". For RQ4, we hence only investigate the effectiveness of CBUA and PMT in finding "covered but alive mutants". In particular, in order to provide actionable feedback to the developers, we control for types of mutants when investigating RQ4. The answer to RQ4 enables us to get a clear understanding on the practical benefits of using the one approach over the other.

4.3 Data Sets

In this study, we use 163 O-PMT "*PIT*" data sets and 650 Ext-PMT "*PIT*" data sets to investigate RQ1 and RQ3[1]:

- *163 O-PMT "PIT" data sets.* This data sets shared online by [13], were collected from 163 real-word Java projects (9 basic projects + 154 additional projects). For each of the 163 projects, Zhang *et al.* first employed the widely used mutation testing tool *PIT* (with 14 mutation operators, shown in the appendix) [37] to generate the corresponding mutants. Based on the resulting mutants and the associated manually written test suite, they then generated a corresponding data set for each project. In each data set, an instance represents a mutant which consists of: (1) a total of 15 features on the execution, infection, and propagation that may relate to whether a mutant can be killed; and (2) the label indicating whether a mutant is "killed" or "alive" under the whole test suite.
- *650 Ext-PMT "PIT" data sets.* This data sets shared online by [44], were collected from 650 real-word Java projects. For each of the 650 projects, Mao *et al.* first employed the widely used mutation testing tool *PIT* (with 17 mutation operators, shown in the appendix) to generate the corresponding mutants. Based on the resulting mutants and the associated manually written test suite, they then generated a corresponding data set for each project. In each data set, an instance represents a mutant which consists of: (1) a total 95 features (including the execution, infection, and propagation features) that may relate to whether a mutant can be killed; and (2) the label indicating whether a mutant is "killed" or "alive" under the whole test suite.

We use 6 O-PMT "*Major*" data sets, 4 O-PMT "*randoop*" data sets, and the 4 "*PIT-HOM*" data sets to investigate RQ2:

- *6 O-PMT "Major" data sets*. In order to investigate the influence of mutation testing tools, Zhang *et al.* employed another widely used mutation testing tool *Major* (with all the 8 mutation operators) [38] to generate the mutants. For the 9 basic projects, *Major* successfully generated the mutants for *apns, lafj, lang, msg, uaa*, and *wire*. Based on the resulting mutants and the associated manually written test suites, they generated 6 "*Major*" data sets.
- *4 O-PMT "randoop" data sets*. In order to investigate the influence of test types, Zhang *et al.* used the well-known tool *randoop* [39] to generate test cases. For the 9 basic projects, *randoop* successfully generated test cases for 4 projects (i.e., *apns, lafj, lang*, and *msg*). Based on the mutants generated by *PIT* and those automatically generated test suites, they generated 4 "*randoop*" data sets.
- *4 "PIT-HOM" data sets*. In order to investigate the influence of mutant magnitude, we employed the

1. All data sets and the script of CBUA can be found on https://github.com/zhangpengNJU/CBUA.

state-of-the-art high order mutant (HOM) generation tool *PIT-HOM* (with 19 mutation operators, shown in the appendix) [45] to generate the HOMs. A HOM is a program mutated by multiple mutators, which can be seen as a combination of first-order mutants. In this paper, the mutants mentioned before are all first-order mutants. In order to avoid the combination explosion, we only use the second-order operator to generate HOMs. For the 9 basic projects, *PIT-HOM* successfully generated the mutants for 4 projects (i.e., *lafj*, *lang*, *msg*, and *wire*). Based on the mutants generated by *PIT-HOM* and the associated manually written test suites, we generated 4 "*PIT-HOM*" data sets.

We use only 650 Ext-PMT "*PIT*" data sets shared online by [44] to investigate RQ4, since they provide the type of mutation for each mutant. We do not use 163 O-PMT "*PIT*" data sets shared by [13] to investigate RQ4. The reason is that, in 163 O-PMT "*PIT*" data sets, the detailed mutator name corresponding to each mutant is unavailable.

Note that the O-PMT data sets shared by [13] contain data noise (see Section 4.4 for detail). In our study, we first filter out noise instances in each data set and then use the cleaned data sets to investigate RQ1, RQ2, and RQ3. In other words, if an O-PMT data set contains noise instances, our choice is to filter out noise instances rather than exclude the entire data set. This choice enables us to use the data sets from the same projects as used in [13] to conduct a direct comparison between CBUA and PMT. In [44], Mao *et al.* indeed shared 654 Ext-PMT "PIT" data sets. Among the 654 data sets, 4 data sets contain noise instances according to the rules described in Section 4.4 (see Section 4.4 for detail). In our study, we exclude these 4 data sets for analysis (rather than retain them after filtering out noise instances). Consequently, we have 650 quality Ext-PMT "*PIT*" data sets to investigate RQ1, RQ3, and RQ4. Combing the O-PMT data sets and the Ext-PMT data sets together, we are able to conduct an in-depth comparison between CBUA and PMT.

4.4 Data Preprocessing

After manually inspecting the data sets shared by Zhang *et al.* 's study [13] and Mao *et al.*'s study [44], we find that there are a number of noise instances (i.e., mutants). For these noise instances, the relationships between "*numExecuteCovered*" (i.e., the number of times the mutated statement is executed under the whole test suite) and "*numTestCovered*" (i.e., the number of test cases from the test suite covering the mutated statement) are counter-intuitive. Before subsequent analyses, we take the following rules to filter out noise instances:

1) filter out those instances: ($numTestCovered == 0 \wedge numExecuteCovered \neq 0$) $\vee$ ($numTestCovered \neq 0 \wedge numExecuteCovered == 0$). Intuitively, for any instance, if one of *numTestCovered* and *numExecuteCovered* is equal to zero but the other one is not, then it is a noise instance and hence should be filtered out. The reason is as follows: if $numTestCovered == 0$, this means that the mutated statement is not covered by any test case. As a result, the mutated statement should not be executed (i.e., we should have: $numExecuteCovered == 0$); if $numExecuteCovered == 0$, this means that the mutated statement is not executed by the whole test suite. As a result, the mutated statement should not be covered by any test case (i.e., we should have: $numTestCovered == 0$).
2) filter out those instances: $numExecuteCovered == 0 \wedge$ isKilled == "Yes". Intuitively, for any instance, if its *numExecuteCovered* is equal to zero but the label indicates that it is killed by the test suite, then it is a noise instance and hence should be filtered out. The reason is as follows: for any instance, if the mutated statement is not executed under the test suite, the instance must survive (i.e., we should have: isKilled == "No").
3) filter out those instances: $numExecuteCovered < numTestCovered$. Intuitively, for any instance, if its *numExecuteCovered* is less than *numTestCovered*, then it is a noise instance and hence should be filtered out. The reason is as follows: for any instance, if it is covered by a test case, the mutated statement must be executed at least once (i.e., we should have: $numExecuteCovered \geq numTestCovered$).
4) filter out those instances: $numExecuteCovered < 0$. Intuitively, for any instance, if its *numExecuteCovered* is less than 0, then it is a noise instance and hence should be filtered out. The reason is as follows: for any instance, its *numExecuteCovered* must be a non-negative number.

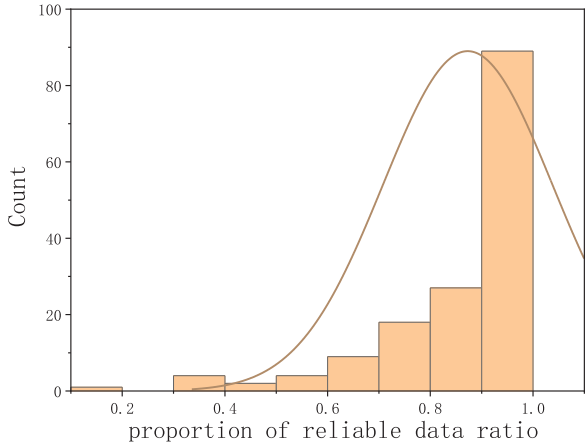


Fig. 3. The retained ratio distribution for 163 O-PMT "*PIT*" projects. This ratio is the size of the filtered data set divided by the size of the original data set. The *x* axis represents the ratio. The *y* axis represents the number of projects falling in specific ranges.

Fig. 3 shows the retained ratio distribution for 163 O-PMT "*PIT*" data sets (a data set is generated for a project). As can be seen, most projects have a retained ratio large than 80 percent. However, we also can see that few projects have a very low retained ratio. In particular, the project "metrics-statsd" has a retained ratio of 18 percent, resulting in only 66 retained instance. Table 2 shows the retained ratio for the O-PMT "*Major*" and O-PMT "*randoop*" data sets, ranging from 75 to 98 percent. In Table 2, the first column shows the name of projects. The second and third columns respectively show the details of retained ratio of *Major* and *randoop*. For Mao *et al.*'s study, 654 projects were used. Only the fourth kind of noise is detected in 4 projects (*FaceGecognition*, *nlp-lang*, *sanselan*, and *scrypt*) out of 654 projects. Considering the high quality of these data sets and the large number of projects, in order to get a reliable comparison

TABLE 2
The Retained Ratio in the 6 O-PMT "*Major*" and 4 O-PMT "*randoop*" Data Sets

	Major			*randoop*		
	Original	Retained	Ratio	Original	Retained	Ratio
apns	846	829	0.980	1101	1027	0.933
lafj	10344	10282	0.994	8585	8413	0.980
lang	21900	20238	0.924	26411	24690	0.935
msg	4841	4538	0.937	7329	7289	0.995
uaa	2568	1951	0.760	-	-	-
wire	2104	1842	0.875	-	-	-

result, we abandon these four projects and hence obtain 650 Ext-PMT "*PIT*" data sets.

Data distribution on the 163 filtered O-PMT "PIT" data sets. As shown in Fig. 4a, for most projects, the number of mutants is less than ten thousand. As shown in Fig. 4b, most projects have a mutation score ranging from 0 and 0.8. As shown in Fig. 4c, many mutants have a *numTestCovered* larger than 500, i.e., the corresponding mutated statements are covered by at least 500 test cases. As shown in Fig. 4d, many mutants have a *numExecuteCovered* larger than 1000, i.e., the corresponding mutated statements are executed at least 1000 times.

Data distribution on the 650 Ext-PMT "PIT" data sets. As shown in Fig. 4a, for most projects, the number of mutants is less than fifty thousand. As shown in Fig. 4b, most projects have a mutation score ranging from 0 to 0.8. As shown in Fig. 4c, many mutants have a *numTestCovered* larger than 1000, i.e., the corresponding mutated statements are covered by at least 1000 test cases. As shown in Fig. 4d, many mutants have a *numExecuteCovered* larger than 1000, i.e., the corresponding mutated statements are executed at least 1000 times.

The influence of noise data on the performance of PMT on the 9 O-PMT basic projects. In [13], Zhang *et al.* reported the detailed prediction performance of O-PMT on the 9 basic projects, including the weighted F_1 and the absolute prediction error of mutation score [13]. It is interesting to know the influence of noise data on the performance of O-PMT. Table 3 summarizes the prediction performance of O-PMT based on the original data and the filtered data. In Table 3, the first column shows the names of projects. The second and third columns respectively show for each project the weighted F_1 and APE on the filtered and original data. As can be seen, after filtering out the noise data, O-PMT exhibits a slightly worse performance. This indicates that filtering out noise data does not have a large influence on the effectiveness of O-PMT.

Possible causes of noise instances in the O-PMT data sets. Zhang *et al.* used Cobertura to collect *numExecuteCovered* and *numTestCovered* [13]. However, it has been reported that Cobertura may produce incorrect coverage information. For example, for the following code:

```
if(a != null){
system.out.print("Hello");
}
```

According to the issue #337 opened by NicolasWei, Cobertura reported that "`if(a != null){`" was not covered but `system.out.print("Hello")` inside was covered [49]. This is a possible reason why many instances exhibit an illogical relationship between *numExecuteCovered* and *numTestCovered* in the O-PMT data sets. Indeed, our previous studies show that it is not uncommon to find incorrect coverage information even for well-known mature code coverage tools such as gcov and llvm-cov [50], [51]. Furthermore, too many executions can result in an overflow, producing a *numExecuteCovered* less than 0. In contrast, Mao *et al.* used their own tool (based on ASM via extending IntelliFL), rather than Cobertura, to collect *numExecuteCovered* and *numTestCovered* [44]. This is the possible reason why 650 of the 654 Ext-PMT data sets contain no noise instances according to the rules described in Section 4.4.

4.5 Performance Evaluation

Evaluation of Mutation Score Prediction. Following the method in [13], we use Absolute Prediction Error (i.e., *APE*) as an indicator to evaluate the effectiveness of mutation score prediction. APE represents the absolute difference between the predicted mutation score and the actual mutation score. The smaller the *APE*, the better the prediction performance.

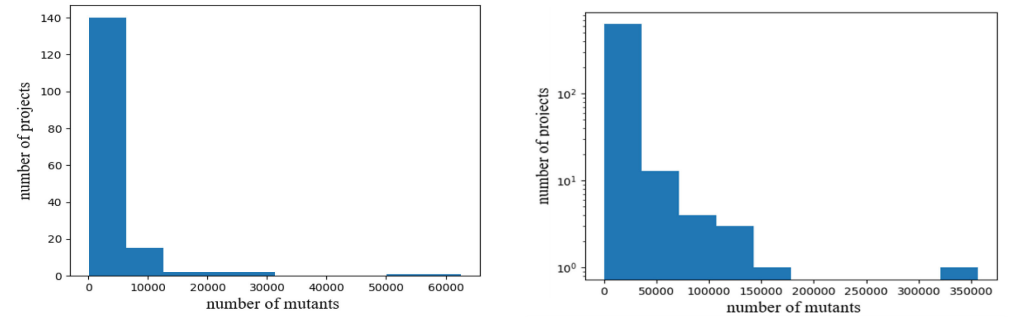
As described in Section 3.3, CBUA uses the estimated number of alive mutants E_{alive} to predict the mutation score (see equation (5)). As can be seen, this process does not involve mutant classification. The computation of its APE is straight-forward: the absolute difference between the predicted mutation score and the actual mutation score.

Unlike CBUA, both PMT and COV use the result of mutant classification to predict a mutation score. Assume that Fig. 5 is the corresponding confusion matrix. Here, A is the number of actually "killed" mutants classified as "killed", B is the number of actually "killed" mutants classified as "alive", C is the number of actually "alive" mutants classified as "killed", and D is the number of actually "alive" mutants classified as "alive". Recall that N is the total number of mutants, in which n_1 mutants are actually killed and n_2 mutants are actually alive. Therefore, we have: $A + B + C + D = N$, $A + B = n_1$, and $C + D = n_2$. On the one hand, according to this confusion matrix, the predicted mutation score is:

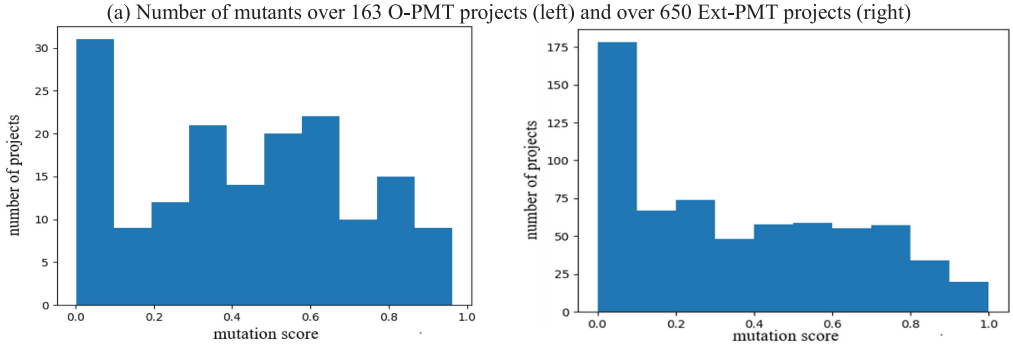
$$pms = \frac{A + C}{N}.$$

On the other hand, the actual mutation score is:

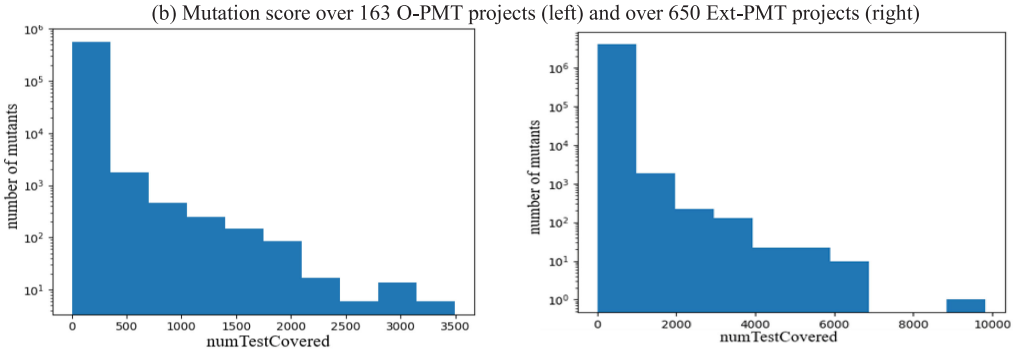
$$ams = \frac{n_1}{N}.$$



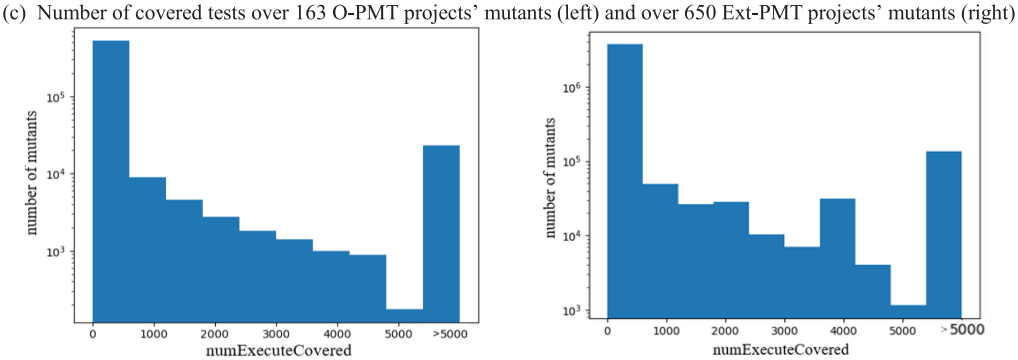
(a) Number of mutants over 163 O-PMT projects (left) and over 650 Ext-PMT projects (right)



(b) Mutation score over 163 O-PMT projects (left) and over 650 Ext-PMT projects (right)



(c) Number of covered tests over 163 O-PMT projects' mutants (left) and over 650 Ext-PMT projects' mutants (right)



(d) Number of executions over 163 O-PMT projects' mutants (left) and over 650 Ext-PMT projects' mutants (right)

Fig. 4. The distribution of features used in this paper.

As such, the absolute prediction error is:

$$APE = |pms - ams| = \left|\frac{A + C - n_1}{N}\right|.$$

Evaluation of Mutant Label Prediction. In mutation label prediction, a mutation is assigned with one of the following two types of labels: "killed" or "alive". When evaluating mutant label prediction, should we concentrate on the effectiveness of "killed" or "alive" mutant prediction? On the one hand, information about killed mutants is the basis to compute a mutation score for evaluating test suite effectiveness (for example, for the baseline approaches such as PMT and COV). On the other hand, information about alive mutants is the basis to identify "the weaknesses in the current test suite and to add new test cases accordingly" [13]. In other words, killed mutant prediction is mainly used to evaluate test suite effectiveness, while alive mutant prediction is mainly used to guide test suite enhancement. As mentioned above, we use APE to evaluate the effectiveness of mutation score prediction, which is influenced by the effectiveness of killed mutant prediction. In this sense, the effectiveness of killed mutant prediction has already been implicitly taken into account in the evaluation of mutation score prediction.

TABLE 3
The Prediction Performance of O-PMT on the Original and Filtered O-PMT Data: the 9 "*PIT*" Basic Projects

	Weighted F_1 of O-PMT			Absolute Prediction Error of O-PMT		
	Filtered data	Original data	Difference	Filtered data	Original data	Difference
apns	0.847	0.903	–0.056	0.111	0.078	+0.033
assj	0.955	0.934	+0.021	0.055	0.037	+0.018
joda	0.896	0.911	–0.015	0.133	0.083	+0.050
lafj	0.899	0.871	+0.028	0.132	0.106	+0.026
lang	0.915	0.906	+0.009	0.084	0.062	+0.022
msg	0.901	0.925	–0.024	0.098	0.068	+0.030
uaa	0.830	0.935	–0.105	0.117	0.052	+0.065
vrapt	0.928	0.910	+0.018	0.071	0.067	+0.004
wire	0.845	0.885	–0.040	0.005	0.003	+0.002
Average	0.891	0.909	–0.018	0.090	0.064	+0.026

Therefore, in the following part, when evaluating mutant label prediction, we concentrate on the effectiveness of "alive" mutant prediction. That is to say, in our study, an alive mutant is regarded as a positive instance, while a killed mutant is regarded as a negative instance. Ideally, a good mutant label prediction approach should lead to a high accuracy on mutation score prediction as well as on alive mutant prediction.

We use F_1 and AUC as the indicators to evaluate the effectiveness of mutant label prediction. F_1 is the harmonic mean of precision P and recall R:

$$F_1 = \frac{2 \times P \times R}{P + R}.$$

Here, P is the fraction of instances classified as positive that really are positive, i.e.,

$$P = \frac{D}{B + D},$$

R is the fraction of positive instances that are correctly identified positive, i.e.,

$$R = \frac{D}{C + D}.$$

Consequently, we have:

$$F_1 = \frac{2D}{2D + B + C}.$$

AUC is the area under ROC curve, measuring the overall discrimination ability of a classifier (A high AUC indicates a better prediction effectiveness). Compared with F_1, AUC has the following advantages: (1) it does not depend on the classification threshold; (2) it is invariant to the prior probabilities of positive and negative observations; and (3) it can be interpreted as the probability that a randomly chosen positive instance is predicted with the greater likelihood of being positive than a randomly chosen negative instance.

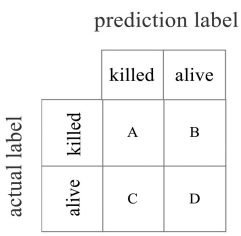


Fig. 5. The confusion matrix.

Evaluation of Predictability. For each mutant label prediction, both PMT and CBUA can provide a predictability value to quantify the confidence of the prediction by the classifier. Since misclassification may happen, developers should not take it granted that a high predictability indicates an accurate prediction. In this study, we aim to evaluate the trustworthiness of mutant predictability, i.e., the validity that a high predictability indicates a more accurate prediction. For a classifier, if it produces a highly trustworthy predictability, then it is logical that developers use mutant predictability to "choose to either believe in the predictive result when the predictability is high, or to doubt the predictive result and run the mutant against the tests to get more accurate results when the predictability is low." [13]. Intuitively, it is natural to use the following idea to measure the trustworthiness of mutant predictability: if the prediction is correct, it is highly probable that a higher (lower) predictability indicates a more (less) accurate prediction; if the prediction is incorrect, it is highly probable that a higher (lower) predictability indicates a less (more) accurate prediction. Numerically, for each mutant predictability, we define its trustworthiness as:

$$Trustworthiness = \begin{cases} predictability & if\ correct \\ 0.5 - predictability & if\ incorrect \end{cases}.$$

Since predictability has a range between 0 and 0.5, trustworthiness also has a range between 0 and 0.5. A larger value indicates a higher trustworthiness.

Evaluation of Practical Benefits. For developers, the most useful information is to know those mutants covered but alive (i.e., not killed) by a test suite. Such information could help them understand the weaknesses in the current test suite and hence design new test cases to enhance the test suite. As described in previous sections, both CUBA and PMT predict the surviving mutants, which may include uncovered mutants (they are certainly alive), "covered but alive mutants", and "covered and killed mutants". Given CBUA or PMT, how to evaluate its cost-benefit in helping identify "covered but alive mutants"? To this end, we

should filter out all the uncovered mutants from the predicted surviving mutants before analysis. Without loss of generalization, for a target project, assume that there are n covered mutants in total, of which k mutants are alive. For a prediction approach m, assume that, of the predicted surviving mutants, there are x covered mutants, in which y mutants are alive. In order to evaluate the cost-benefit of the prediction approach m, we make the following assumptions

- All the x covered mutants are inspected;
- The inspection is perfect, i.e., all the y alive mutants in the inspected x mutants are found;
- For computing a benefit, we use the random selection model (mutants to be inspected are selected from the n covered mutants by chance) as a baseline of comparison.

As a result, when using the prediction approach m, the percentage of covered mutants needed to be inspected is $x/n \times 100$ percent, while the percentage of "covered but alive mutants" found is $y/k \times 100$ percent. Similar to [47], [48], we evaluate how effective the prediction model m is in reducing the effort for inspection compared to a random model that achieves the same recall of "covered but alive mutants". To this end, we use the following ER (effort reduction) indicator:

$$ER = \frac{effort(random) - effort(m)}{effort(random)}.$$

Here, effort(m) $= x/n \times 100$ percent and effort(*random*) $= y/k \times 100$ percent. In other words, we have:

$$ER = \frac{\frac{y}{k} - \frac{x}{n}}{\frac{y}{k}}.$$

A positive ER indicates that the prediction approach m is superior to a random model for finding "covered but alive mutants". A higher ER indicates a better cost-benefit.

Examination of Statistical Significance and Practical Importance of the Difference. When comparing CBUA against a baseline approach, the following problems naturally arise: is their difference statistically significant? Is their difference practically important? In our study, we use the Wilcoxon signed-rank test to examine whether the difference is statistically significant at the significance level of 0.05 [40]. However, as stated in [52], with large enough experimental runs (i.e., large enough target projects in our study), "we can obtain statistically difference even if that difference is so small as to be of no practical value". Therefore, "it is in addition crucial to assess the magnitude of the improvement" [52]. In this context, effect size, a statistical value, is often used to measure the magnitude of the difference. In our study, we employ the Cliff's δ [41], which is used for median comparison on effect size, to examine whether the magnitude of the difference between the performances of two approaches is important from the viewpoint of practical application. Specifically, for two samples $X = \{x_1, \ldots, x_m\}$ and $Y = \{y_1, \ldots, y_n\}$, the Cliff's δ is defined as:

$$\delta = \frac{1}{mn}\sum_{i=1}^{m}\sum_{j=1}^{n} sign(i,j),$$

where the sign function is defined as follows:

$$sign(i,j) = \begin{cases} +1 & x_i > y_j \\ -1 & x_i < y_j \\ 0 & x_i = y_j \end{cases}.$$

By convention, the magnitude of the difference is considered either trivial ($|\delta| < 0.147$), small ($0.147 \leq |\delta| \leq 0.33$), moderate ($0.33 \leq |\delta| \leq 0.474$), or large ($|\delta| > 0.474$). For a given performance indicator, if the effect size between two approaches is trivial, it means that, under this indicator, even if the difference between the two approaches is statistically significant, a simple approach would be preferred from the viewpoint of practical use.

4.6 Data Analysis Method

RQ1: We use 163 O-PMT "*PIT*" data sets (from 9 basic projects and 154 additional projects) in [13] and 650 Ext-PMT "*PIT*" data sets in [44] to investigate the effectiveness of CBUA under the following four scenarios:

- *Scenario 1: Many-to-many (MTM) prediction from 9 O-PMT basic projects to 154 O-PMT additional projects*. Under this scenario, the data sets from 9 basic projects are combined as the training data, while the data sets from 154 additional projects are used as the test data sets. We refer to this setting as "9→154". For each approach, we have 1 experiment under this setting. Note that O-PMT needs the training data but CBUA and COV do not need the training data. After obtaining the prediction results on 154 additional projects, we can investigate the effectiveness of CBUA compared with O-PMT and COV.
- *Scenario 2: Leave-one-out (LOO) prediction among 163 O-PMT projects*. Under this scenario, the data sets from all the 163 projects are used to conduct the LOO prediction. In other words, each data set is used as the test data set and the other 162 data sets are combined as the training data. We refer to this setting as "162→1". For each approach, we have 163 experiments in this setting. After obtaining the results on 163 projects, we can investigate the effectiveness of CBUA compared with O-PMT and COV.
- *Scenario 3: One-to-one (OTO) prediction among 163 O-PMT projects*. Under this scenario, the data sets from all the 163 projects are used: one data set used as the training data and another one data set used as the testing data. We refer to this setting as "1→1". For each approach, we have 163 experiments under this setting. After obtaining the results on 163 projects, we can investigate the effectiveness of CBUA compared with O-PMT and COV.
- *Scenario 4: 5-fold cross validation prediction among 650 Ext-PMT projects*. Under this scenario, the data sets from all the 650 projects are used to conduct the 5-fold cross validation prediction. In other words, each fold is used as the test data set with 130 projects and the other 4 folds are combined as the training data. We refer to this setting as "520→130". For each approach, we have 650 experiments in this setting. After obtaining the results on 650 projects, we can

investigate the effectiveness of CBUA compared with Ext-PMT and COV.

Note that scenario 1 ($''9 \rightarrow 154''$) is used in Zhang *et al.*'s study [13], we keep them in our study to enable a direct comparison between our results and their results. Scenario 2 ($''162 \rightarrow 1''$) aims to simulate the scenario that rich training data are available, while scenario 3 ($''1 \rightarrow 1''$) aims to simulate the scenario that only limited training data are available. Scenario 4 ($''530 \rightarrow 120''$) aims to obtain the comparison results without noise instances. Following in [44], we use 5-fold cross-validation prediction in scenario 4. Note that, due to the intensive computation of Ext-PMT, we do not apply LOO and OTO to the 650 Ext-PMT data sets. With the four scenarios, we can make an in-depth understanding of the effectiveness of CBUA.

RQ2: We use 6 O-PMT "*Major*" data sets to investigate the influence of mutation testing tools, 4 O-PMT "*randoop*" data sets to investigate the influence of testing types, and use 4 "*PIT-HOM*" data sets to investigate the influence of the order of mutants. More specifically, we use the same method as used in Zhang *et al.*'s study [13] to conduct the analysis:

- *Influence of mutation testing tools*. Each "*Major*" data set is used as the test data set to evaluate the influence. For CBUA and COV, they are directly applied to the test data to obtain the prediction results. For O-PMT, a supervised model is first trained using the training data combined from the remaining five "*Major*" data sets.
- *Influence of testing types*. Each "*randoop*" data set is used as the test data set to evaluate the influence. Again, for CBUA and COV, they are directly applied to the test data to obtain the prediction results. For O-PMT, a supervised model is first trained using the training data combined from the "*PIT*" data sets corresponding to the 9 basic projects without the target project.

Besides [13], we conduct the analysis:

- *Influence of mutant magnitude*. Each "*PIT-HOM*" data set is used as the test data set to evaluate the influence. For CBUA and COV, they are directly applied to the test data to obtain the prediction results. For O-PMT, a supervised model is first trained using the training data combined from the remaining 3 "*PIT-HOM*" data sets.

With the above experimental settings, we aim to investigate what the influence of different mutation testing tools, test types and mutation magnitude on CBUA is, especially compared with O-PMT and COV.

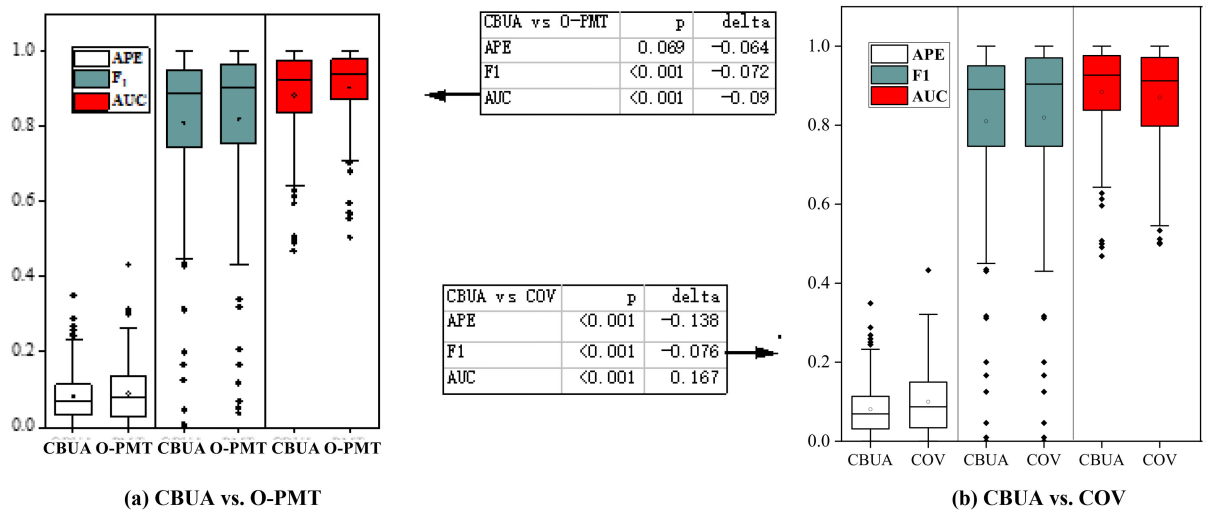
RQ3: We use 163 O-PMT "*PIT*" data sets and 650 Ext-PMT "*PIT*" data sets to investigate mutant predictability and its trustworthiness. On the one hand, we use a heatmap to visualize the distribution of mutant predictability under CBUA. In this heatmap, columns indicate the projects, while rows indicate the 10 intervals ranging from 0 to 0.5 with a step length of 0.05. The value in each cell represents the percentage of mutants whose predictability lies in the corresponding interval. Such a heatmap allows us to understand how many mutants have a high predictability. On the other hand, we use another heatmap to visualize the distribution of the trustworthiness of mutant predictability under CBUA. Furthermore, we examine if CBUA has a higher trustworthiness compared with O-PMT/Ext-PMT by the following ways:

- A global basis. First, combine the mutants in all projects to obtain a global mutant set. Second, for each mutant in the set, generate the corresponding trustworthiness values of predictability under CBUA and O-PMT/Ext-PMT, respectively. Third, compute the percentage of mutants that CBUA has a higher trustworthiness and the percentage of mutants that O-PMT/Ext-PMT has a higher trustworthiness. Fourth, employ one-sample t-test to examine whether there is a significant difference between these two percentages. Furthermore, we employ the Cohen's h to examine whether the magnitude of the difference between these two percentages are important from the viewpoint of practical application. By convention, the magnitude of the difference is considered either trivial ($|h| < 0.2$), small ($0.2 \leq |h| \leq 0.5$), moderate ($0.5 \leq |h| \leq 0.8$), or large ($|h| > 0.8$) [42].
- A per-project basis. First, use a scatter point (the x axis indicates each project and the y axis the indicates percentage) to report the percentage of mutants that CBUA has a higher trustworthiness and the percentage of mutants that O-PMT/Ext-PMT has a higher trustworthiness within each project. Second, use boxplots to compare the distributions of mean/median trustworthiness on each project for CBUA and O-PMT/Ext-PMT. In either case, employ the Wilcoxon signed-rank test to examine the statistical significance and the Cliff's delta to examine the practical importance for their difference.

With the above analysis, we aim to understand the characteristics of mutation predictability under CBUA, especially whether CBUA have a more trustworthy predictability than O-PMT/Ext-PMT.

RQ4: We use 650 Ext-PMT "*PIT*" data sets to investigate the practical benefits of CBUA. In our study, we investigate the effectiveness of CBUA in finding "covered but alive mutants" using the following two methods: "alive" mutant prediction guided and mutant predictability guided.

- "Alive" mutant prediction guided. For a given test suite on a target project, we first apply CBUA to classify the corresponding mutants into two groups (according to their survival probability): predicted "alive" and predicted "killed". Then, we exclude all uncovered mutants from the predicted "alive" group. Finally, the remaining predicted "alive" mutants are recommended to developers for inspection. The process of applying Ext-PMT to help find "covered but alive mutants" is similar. In order to make an objective comparison, we adjust the classification threshold of Ext-PMT such that Ext-PMT has the same recall as CBUA. This enables to compare their incurred inspection effort when finding the same number of "covered but alive mutants". In particular, in order to control for the confounding effect of mutator types, we report their comparison results (i.e., ERs) over different mutators in addition to the overall result.



(a) CBUA vs. O-PMT (b) CBUA vs. COV

Fig. 6. The effectiveness of CBUA and its comparison with O-PMT and COV under scenario 1: 9→154.

- Mutant predictability guided. According to [13], a low predictability indicates a low confidence for prediction. As a result, it is a natural practice to inspect these mutants having low predictability first [13]. For a given test suite on a target project, we recommend those mutants with a predictability less than the threshold r to developers for inspection. In this context, CBUA and Ext-PMT will lead to two different sets of low-predictability mutants for inspection. Again, in order to control for the confounding effect of mutator types, we report their comparison results (i.e., ERs) over different mutators in addition to the overall result. In particular, we change r from 0.05 to 0.30 to investigate the influence of the predictability threshold.

In practice, after mutant label prediction, it is natural to use the predicted "alive" mutants to guide the finding of "covered but alive mutants". By mutant predictability, we are able to provide additional means to help developers find "covered but alive mutants". In this sense, for identifying "covered but alive mutants", the former is the main scenario, while the latter is a supplementary scenario. With the above experimental settings, we aim to investigate the practical benefits of CBUA, especially when compared with Ext-PMT.

5 Experimental Results

In this section, we report in detail the results on the effectiveness of CBUA, as well as its comparison with the state-of-the-art supervised approach (i.e., O-PMT and Ext-PMT) and the simple approach (i.e., COV).

5.1 RQ1: Effectiveness of CBUA

We analyze the effectiveness of CBUA under four scenarios: MTM from 9 O-PMT basic projects to 154 O-PMT additional projects (9→154), LOO among 163 O-PMT projects (163→1), OTO among 163 O-PMT projects (1→1), and 5-fold cross validation prediction among 650 Ext-PMT projects (520→130).

Scenario 1: MTM from 9 O-PMT basic projects to 154 O-PMT additional projects. Fig. 6 shows the results under scenario 1 "9→154". On the 154 additional projects, on average, CBUA achieves an APE value of 0.081, an F_1 value of 0.810, and an AUC value of 0.884.

- CBUA vs. O-PMT. In terms of APE, CBUA has a lower (better) average value than O-PMT (0.081 vs. 0.090). The p-value indicates an approximately significant difference, while the Cliff's delta indicates a trivial effect size. In terms of both F_1 and AUC, CBUA is significantly lower, but the effect size is trivial. The above experimental results reveal that, CBUA has a comparable prediction effectiveness with O-PMT on the 154 additional projects.
- CBUA vs. COV. In terms of APE, CBUA has a significantly lower value than COV and the effect size is close to non-trivial. This result is as expected (as analyzed in Section 3.6): increasing the number of predicted alive mutant from X in COV to E_{alive} in CBUA will lead to a decrease of APE. In terms of F_1, CBUA exhibits a significantly lower value but the effect size is trivial. In terms of AUC, CBUA is significantly better than COV and the effect size is non-trivial. The above experimental results reveal that CBUA is better than COV in both mutation score prediction and mutant classification.

Combining the above results, we can see that, in the first scenario: on the one hand, CBUA is comparable to O-PMT in mutation score prediction and mutant classification; on the other hand, CBUA is superior to COV in both mutation score prediction and mutant classification.

Scenario 2: LOO among 163 O-PMT projects. Fig. 7 shows the results under scenario 2 "162→1". On the 163 projects, on average, CBUA achieves an APE value of 0.081, an F_1 value of 0.807, and an AUC value of 0.882.

- CBUA vs. O-PMT. In terms of APE, CBUA has a lower (better) average value than O-PMT (0.081 vs. 0.082). The p-value indicates a non-significant difference, while the Cliff's delta indicates a trivial effect size. In terms of both F_1 and AUC, CBUA is significantly lower, but the effect size is trivial. In order to further understand their difference in mutation score prediction, we use Table 4 to describe the distributions of their APEs. In Table 4, the first column shows the

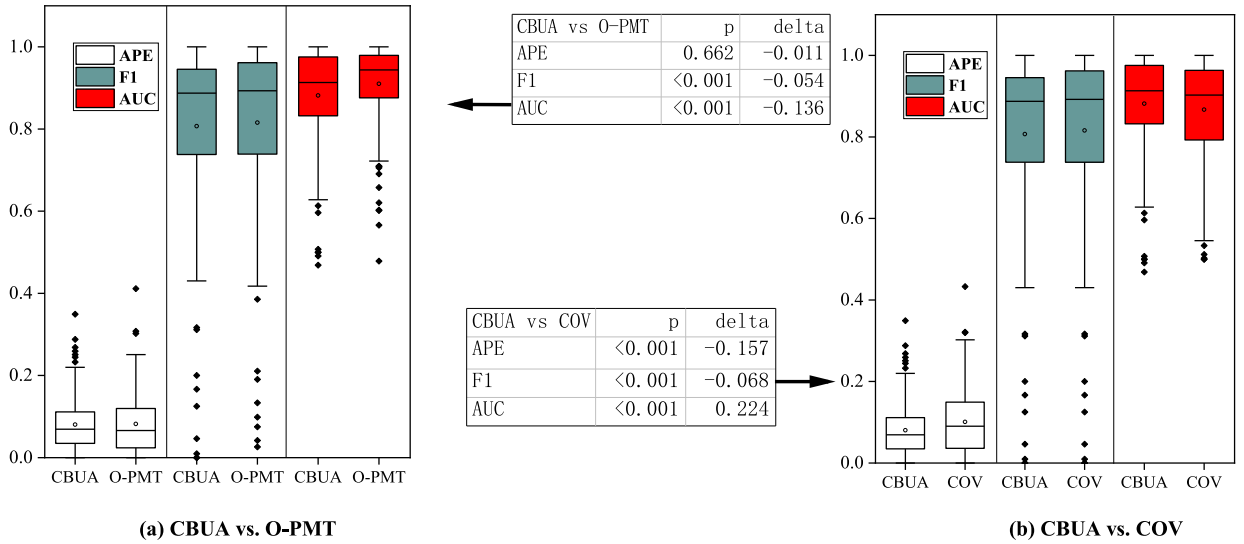


Fig. 7. The effectiveness of CBUA and its comparison with O-PMT and COV under scenario 2: 162→1.

name of each approach. The second to fifth columns respectively show the number of projects with APE less than the value in first row. The last column shows the total number of projects. From Table 4, we can find that CBUA is comparable to O-PMT in mutation score prediction. The above experimental results reveal that, CBUA has a comparable prediction effectiveness with O-PMT on the 163 projects.

- CBUA vs. COV. In terms of APE, CBUA has a significantly lower value than COV and the effect size is non-trivial. In terms of F_1, CBUA exhibits a significantly lower value but the effect size is trivial. In terms of AUC, CBUA is significantly better than COV and the effect size is non-trivial. The above experimental results reveal that CBUA is better than COV in both mutation score prediction and mutant classification.

Combining the above results, we can see that, in the second scenario: on the one hand, although a costly training set is used for O-PMT, CBUA is still comparable to O-PMT in mutation score prediction and mutant classification; on the other hand, CBUA is superior to COV in both mutation score prediction and mutant classification.

Scenario 3: OTO among 163 O-PMT projects. Fig. 8 shows the results under scenario 3 "1→1". On the 163 projects, on average, CBUA achieves an APE value of 0.081, an F_1 value of 0.807, and an AUC value of 0.882. We only compare CBUA with O-PMT, because the comparison between CBUA and COV is consistent with that in scenario 2.

In terms of APE, CBUA has a lower (better) average value than O-PMT (0.081 vs. 0.124). The p-value indicates a significant difference, while the Cliff's delta indicates a non-trivial effect size. In terms of F_1, CBUA is significantly higher, but the effect size is trivial. In terms of AUC, the p-value indicates a non-significant difference, while the Cliff's delta indicates a trivial effect size. To provide a more concrete comparison between CBUA and O-PMT, Table 5 tabulates for each method the number of projects in different APE ranges. In Table 5, the first column shows the name of each approach. The second to fifth columns respectively show the number of projects with APE less than the value in first row. The last column shows the total number of projects. It can be seen that the number of projects with low APE predicted by CBUA is far larger than that of O-PMT. The results indicate that when limited training data are available, O-PMT is no longer appropriate while CBUA still performs well. The above experimental results reveal that, CBUA has a better prediction effectiveness than O-PMT on the 163 projects. We conjecture that the poor performance of O-PMT is due to the heterogeneity among projects, which is the biggest obstacle when performing cross-project prediction. A more detailed discussion about the limitations of supervised methods is shown in Section 6.2.

Combining the above results, we can see that, in the third scenario: CBUA is superior to O-PMT in both mutation score prediction and mutant classification.

TABLE 4
The Number of Projects in Different Error Ranges Under Scenario 2: 162→1

APE	<=0.01	<=0.05	<=0.010	<=0.015	Total
CBUA	17	61	114	142	163
O-PMT	23	66	107	138	163
COV	19	47	93	124	163

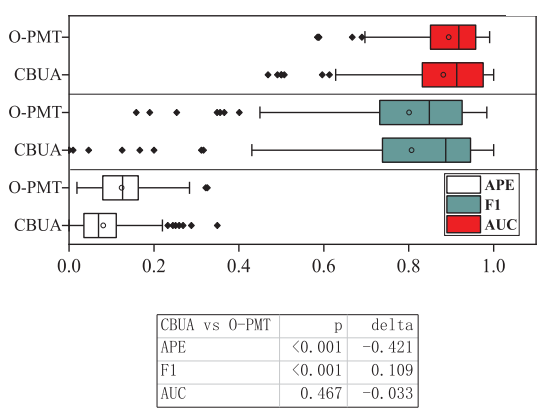


Fig. 8. The effectiveness of CBUA and comparison to O-PMT under scenario 3: 1→1.

TABLE 5
The number of Projects in Different Error Ranges Under Scenario 3: 1→1

APE	<=0.01	<=0.05	<=0.010	<=0.015	Total
CBUA	17	61	114	142	163
O-PMT	0	28	55	113	163
COV	19	47	93	124	163

Scenario 4: 5-fold cross validation prediction among 650 Ext-PMT projects. Fig. 9 shows the results under scenario 4 "520→130". On the 650 projects, on average, CBUA achieves an APE value of 0.120, an F_1 value of 0.773, and an AUC value of 0.880.

- CBUA vs. Ext-PMT. The average value of each evaluation indicator corresponding to CBUA is slightly worse than Ext-PMT. However, it can be found that the effect size is trivial. We find the results of both methods are worse than those of Scenario 2. Our conjecture is that 650 Ext-PMT datasets are of higher quality, and 163 O-PMT datasets lead to the overestimation of both CBUA and PMT. Even Ext-PMT is built with such a large training data set and so many features, CBUA still has a comparable performance.
- CBUA vs. COV. In terms of APE, CBUA has a significantly lower value than COV and the effect size is non-trivial. In terms of F_1, CBUA is not significantly different from COV and the effect size is trivial. In terms of AUC, CBUA is significantly better than COV and the effect size is non-trivial. The above experimental results reveal that overall CBUA is better than COV in both mutation score prediction and mutant classification.

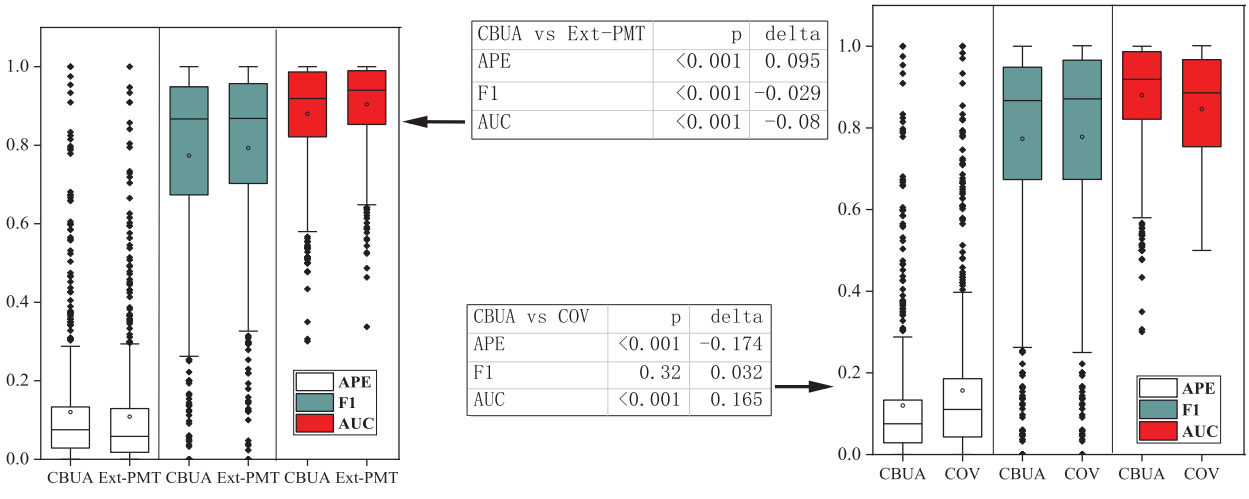
Combining the above results, we can see that, in the fourth scenario: on the one hand, although a costly training set is used for Ext-PMT, CBUA is still comparable to Ext-PMT in mutation score prediction and mutant classification; on the other hand, CBUA is superior to COV in both mutation score prediction and mutant classification.

To conclude, CBUA can provide an effective prediction. On the one hand, CBUA is competitive to PMT when rich training data are available and is superior to PMT when limited training data are available. On the other hand, although CBUA and COV are similar in computation cost, CBUA is superior to COV in both mutation score prediction and mutant classification.

5.2 RQ2: Influence of Application Factors

Influence of mutation testing tools. Fig. 10 shows the effectiveness of CBUA as well as O-PMT and COV on the O-PMT "*PIT*" data sets (i.e., the mutants generated by the PIT tool) and the O-PMT "*Major*" data sets (i.e., the mutants generated by the Major tool). Table 6 summarizes their effectiveness on the O-PMT "*Major*" data sets. In Table 6, the first column shows the name of each project. The second, third, and fourth columns respectively show the APE, F_1, and AUC of three approaches. The last two rows show the p-value and the Cliff's δ. From them, we have the following observations:

- CBUA has a similar prediction effectiveness on the O-PMT "*PIT*" and "*Major*" data sets (see the first column in Fig. 10). On the "*PIT*" data sets, CBUA has an average APE of 0.078, an average F_1 of 0.770, and an average AUC of 0.848. On the "*Major*" data sets, CBUA has an average APE of 0.127, an average F_1 of 0.734, and an average AUC of 0.856. Overall, in terms of each indicator, the magnitude of the difference is not large.
- CBUA exhibits a change trend in prediction effectiveness similar to O-PMT and COV (see each row in Fig. 10). In most cases, for each project, if CBUA has a higher (lower) prediction effectiveness on the "*Major*" data set than that on the "*PIT*" data set, O-PMT and COV also have a higher (lower) effectiveness on the "*Major*" data set. This observation is true for APE, F_1, and AUC.



(a) CBUA vs. Ext-PMT (b) CBUA vs. COV

Fig. 9. The effectiveness of CBUA and its comparison with Ext-PMT and COV under scenario 4: 520→130.

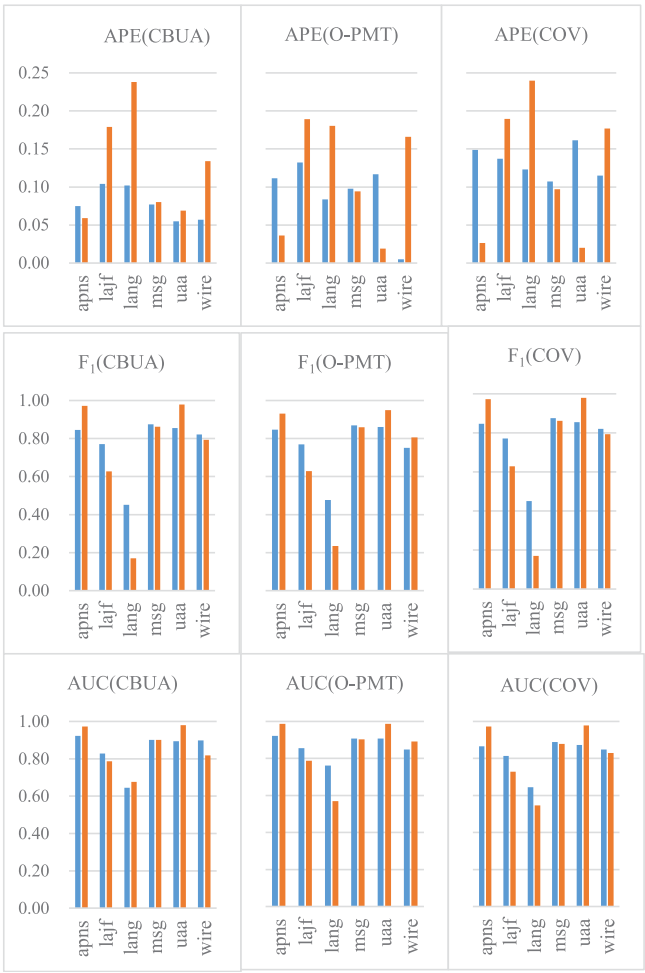


Fig. 10. The effectiveness of three approaches on *PIT* (i.e., shown by the blue bars) and *Major* (i.e., shown by the orange bars). CBUA changes least among the three approaches.

- CBUA achieves a very competitive effectiveness compared with O-PMT and COV on the "*Major*" data sets (see Table 6). In terms of each performance indicator, there is no statistically significant difference between CBUA and O-PMT and between CBUA and COV. Under almost all the cases, the effect size is trivial.

The above observations reveal that CBUA still performs well compared with O-PMT and COV even if a different mutation testing tool is used to generate the mutants.

Influence of testing types. Fig. 11 shows the effectiveness of CBUA as well as O-PMT and COV on the O-PMT "*PIT*" data sets (i.e., manually written test cases) and the "*randoop*" data sets (i.e., automatically generated test cases). Table 7 summarizes their effectiveness on the "*randoop*" data sets In Table 7, the first column shows project name. The second, third, and fourth columns respectively show the APE, F_1, and AUC of three approaches. The last two rows show the p-value and the Cliff's δ. From them, we have the following observations:

- CBUA has a slightly better prediction effectiveness on the "*randoop*" data sets (see the first column in Fig. 11). On the "*PIT*" data sets, CBUA has an average APE of 0.090, an average F_1 of 0.736, and an average AUC of 0.824. On the "*randoop*" data sets, CBUA has an average APE of 0.040, an average F_1 of 0.932, and an average AUC of 0.902. Overall, on the "*randoop*" data sets, CBUA has a better APE, F_1, and AUC.
- CBUA exhibits a change trend in prediction effectiveness similar to O-PMT and COV for F_1 and AUC (see each row in Fig. 11). In terms of F_1 and AUC, for each project, if CBUA has a higher (lower) prediction effectiveness on the "*randoop*" data set than that on the "*PIT*" data set, O-PMT and COV also have a higher (lower) effectiveness on the "*randoop*" data set.
- CBUA achieves a competitive effectiveness compared with O-PMT and COV on the "*randoop*" data sets (see Table 7). Table 7 reports the statistical significance and effect size when comparing CBUA with the other two approaches. Since the sample size is only four, it is of little importance to examine such statistics. Nonetheless, we can see that CBUA has a lower mean APE while maintaining similar mean F_1 and mean AUC, indicating a competitive effectiveness

The above observations reveal that CBUA still performs well compared with O-PMT and COV even if it is used to predict the effectiveness of automatically generated test cases by "*randoop*".

Influence of mutation magnitude. Table 8 summarizes the effectiveness of CBUA as well as O-PMT and COV on the "*PIT-HOM*" data sets. In Table 8, the first column shows the project name. The second, third, and fourth columns, respectively show the APE, F_1, and AUC of three approaches. The last two rows show the p-value and the Cliff's δ. In this

TABLE 6
The Effectiveness on the O-PMT "*Major*" Data Sets

Project	APE			F_1			AUC		
	CBUA	O-PMT	COV	CBUA	O-PMT	COV	CBUA	O-PMT	COV
apns	0.059	0.036	0.027	0.972	0.931	0.972	0.973	0.987	0.973
lafj	0.179	0.189	0.190	0.628	0.629	0.628	0.787	0.788	0.729
lang	0.238	0.180	0.240	0.170	0.235	0.170	0.677	0.569	0.547
msg	0.080	0.094	0.097	0.862	0.860	0.862	0.901	0.903	0.879
uaa	0.069	0.019	0.020	0.979	0.949	0.979	0.980	0.987	0.979
wire	0.134	0.0166	0.177	0.793	0.806	0.793	0.818	0.891	0.829
avg.	0.127	0.0144	0.125	0.734	0.735	0.734	0.856	0.855	0.822
CBUA versus O-PMT	p=0.562, delta=0.056 (trivial)			p=1, delta=0.056 (trivial)			p=0.438, delta=-0.167 (small)		
CBUA versus COV	p=1, delta=0 (trivial)			p=1, delta=0 (trivial)			p=0.156, delta=0.083 (trivial)		

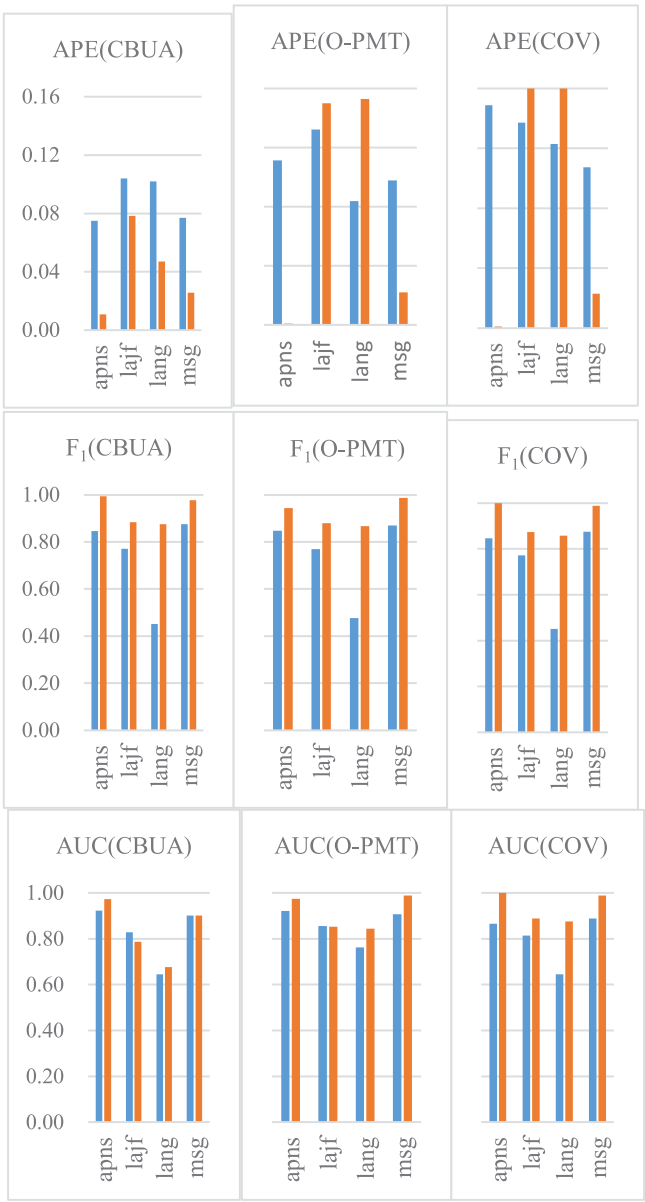


Fig. 11. The effectiveness of three approaches when predicting the execution results regarding manually written (i.e., shown by the blue bars) and automatically generated (i.e., shown by the orange bars) tests. CBUA changes least among the three approaches.

table, the statistical significance and effect size are reported when comparing CBUA with the other two approaches. Since the sample size is only four, it is of little importance to examine such statistics. Nonetheless, we can see that CBUA has a lower mean APE while maintaining similar mean F_1 and mean AUC, indicating a competitive effectiveness.

5.3 RQ3: Mutant Predictability and Its Trustworthiness

Distribution of mutant predictability. Fig. 12 shows for CBUA the distribution of mutant predictability over 163 O-PMT projects. As can be seen, for most columns, the cell in the topmost row is filled with the red color. This indicates that, under CBUA, most mutants have a high mutant predictability. In other words, CBUA has a high confidence in classifying most mutants into killed or alive. Similar phenomena can be observed on the 650 Ext-PMT projects (in order to save space, we do not report the heatmap here. Please see appendix for detail).

Trustworthiness of Mutant Predictability. Fig. 13 shows for CBUA the distribution of the trustworthiness of mutant predictability over 163 O-PMT projects. As can be seen, for most columns, the cell in the topmost row is filled with red color. This indicates that, under CBUA, most mutants have a highly trustworthy predictability. In other words, under CBUA, it is trustworthy that a high mutant predictability indicates a more accurate prediction. Similar phenomena can be observed on the 650 Ext-PMT projects (see appendix for detail).

Comparison With O-PMT/Ext-PMT on a Global basis. The total number of mutants on 163 projects is around 560 thousand. Of these mutants, CBUA has a higher trustworthiness on 30.6 percent of mutants and O-PMT has a higher trustworthiness on 19.3 percent of mutants. By one sample t-test, we obtain $p < 0.001$. This means that CBUA has a significantly higher trustworthiness of mutant predictability than O-PMT. The Cohen's h is 0.262, indicating a non-trivial effect size.

> To conclude, CBUA performs well compared with PMT and COV under different mutation testing tools, different testing types, and different mutation magnitude. In other words, CBUA has a good generalizability.

The total number of mutants on 650 projects is around 4M. Of these mutants, CBUA has a higher trustworthiness on 78.5 percent of mutants and Ext-PMT has a higher trustworthiness on 12.6 percent of mutants. By one sample t-test, we obtain $p < 0.001$. This means that CBUA has a significantly higher trustworthiness of mutant predictability than Ext-PMT. The Cohen's h is 0.667, indicating a large effect size.

Comparison With O-PMT/Ext-PMT on a Per-Project Basis. Fig. 14 shows the winning percentage comparison between CBUA and O-PMT/Ext-PMT. As can be seen, for most projects, CBUA has a higher winning percentage. For (a), the p-value from the Wilcoxon signed-rank test is less than 0.001, indicating that CBUA is significantly better. The Cliff's δ is 0.335, indicating a non-trivial effect size. For (b), the p-value from the Wilcoxon signed-rank test is less than 0.001, indicating that CBUA is significantly better. The Cliff's δ is 0.922, indicating a large effect size. Fig. 15 uses boxplots to visualize the distributions of mean/median trustworthiness values on the 163 O-PMT and 650 Ext-PMT projects. As can be seen, in terms of mean (median) trustworthiness, CBUA has a significantly better than O-PMT/Ext-PMT and the effect size is non-trivial.

The above results of the comparisons are very surprising, as we had thought that mutant predictability under PMT would be more trustworthy. The reason is that PMT uses more features than CBUA when classifying mutants into killed or alive. However, the experimental results show that CBUA outperforms PMT in terms of the trustworthiness of mutant predictability. This implies that more feature data

TABLE 7
The Effectiveness on the "*randoop*" Data Sets

Project	APE			F_1			AUC		
	CBUA	O-PMT	COV	CBUA	O-PMT	COV	CBUA	O-PMT	COV
apns	0.011	0.001	0.001	0.994	0.944	1.000	0.981	0.975	1.000
lafj	0.078	0.150	0.163	0.883	0.879	0.874	0.849	0.853	0.889
lang	0.047	0.153	0.192	0.875	0.867	0.858	0.823	0.844	0.875
msg	0.026	0.022	0.023	0.977	0.987	0.988	0.855	0.988	0.988
avg.	0.040	0.082	0.095	0.932	0.919	0.930	0.902	0.915	0.938
CBUA versus O-PMT	p=0.625, delta=0.125 (trivial)			p=0.625, delta=0.125 (trivial)			p=0.375, delta=-0.25 (small)		
CBUA versus COV	p=0.625, delta=0.125 (trivial)			p=0.875, delta=0.125 (trivial)			p=0.125, delta=0.05 (large)		

TABLE 8
The Effectiveness on the "*PIT-HOM*" Data Sets

Project	APE			F_1			AUC		
	CBUA	O-PMT	COV	CBUA	O-PMT	COV	CBUA	O-PMT	COV
apns	0.095	0.108	0.185	0.642	0.581	0.642	0.775	0.570	0.736
lafj	0.056	0.054	0.014	0.880	0.621	0.880	0.950	0.988	0.910
lang	0.006	0.022	0.071	0.847	0.835	0.847	0.883	0.966	0.862
msg	0.042	0.049	0.056	0.420	0.527	0.420	0.575	0.998	0.596
avg.	0.050	0.058	0.081	0.697	0.641	0.697	0.796	0.880	0.776
CBUA versus O-PMT	p=0.25, delta=0.125 (trivial)			p=0.625, delta=0.375 (medium)			p=0.625, delta=-0.5 (large)		
CBUA versus COV	p=0.375, delta=0.313 (trivial)			p=1, delta=0 (trivial)			p=0.267, delta=0.125 (trivial)		

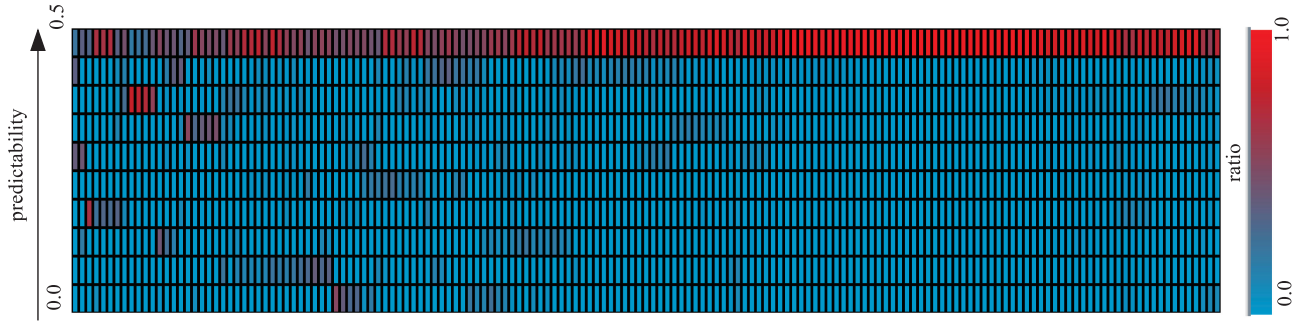


Fig. 12. The distribution of mutant predictability on 163 O-PMT projects. Each column represents a project. The *i*-th cell in each column (from bottom to top) represents the percentage of mutants with predictability between 0.05*(*i*-1) and 0.05**i* in the project

may not necessarily lead to more trustworthy predictability and may confuse both humans and machines.

To conclude, CBUA can provide a practical predictability. On the one hand, for the majority of prediction with high predictability, we are confident that the prediction is correct. On the other hand, CBUA exhibits a more trustworthy predictability than PMT.

5.4 RQ4: Practical Benefits

"Alive" mutant prediction guided "covered but alive mutants" identification. Fig. 16 shows the distribution of covered mutants over 17 mutators for the 650 Ext-PMT "PIT" data sets. As can be seen, for most mutators, around 30 percent of the corresponding covered mutants are alive. In particular, for four mutators, i.e., "MathMutator", "ConditionalsBoundaryMutator", "NakedReceiverMutator",

To conclude, compared with PMT, CBUA has a higher cost-benefit in helping developers find the mutants that are covered but not killed by a test suite.

and "RemoveSwitchMutator", the corresponding covered mutants are more likely to be alive (around 50 percent). Table 9 reports the cost-benefit in finding "covered but alive mutants" under survival probability guided method. In Table 9, the first column shows the name of each mutator. The second and third columns respectively show the total number of mutants and total number of alive mutants for the corresponding mutator. The fourth and fifth columns respectively show the number of inspected, number of detected, and ER of CBUA and Ext-PMT. For CBUA, "Number of inspected" is the number of mutants which are predicted as alive by CBUA among all the covered mutants.

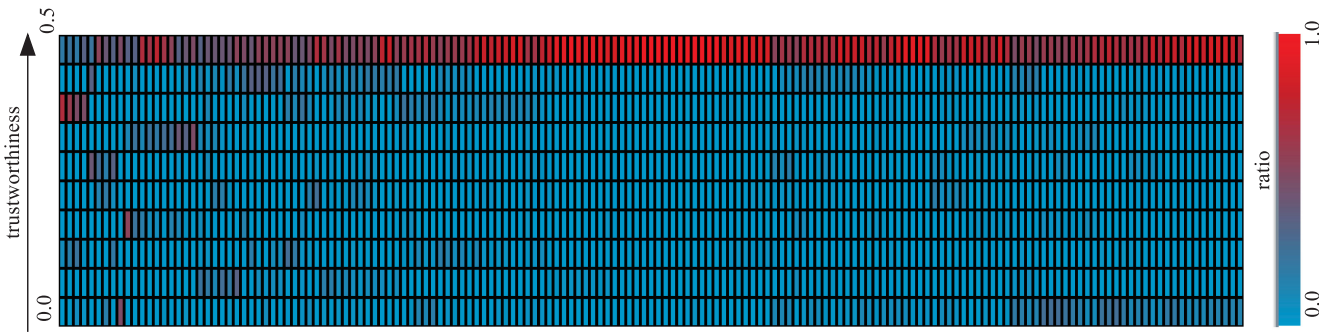
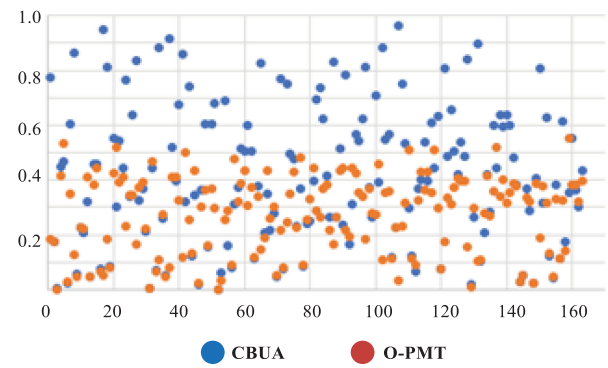


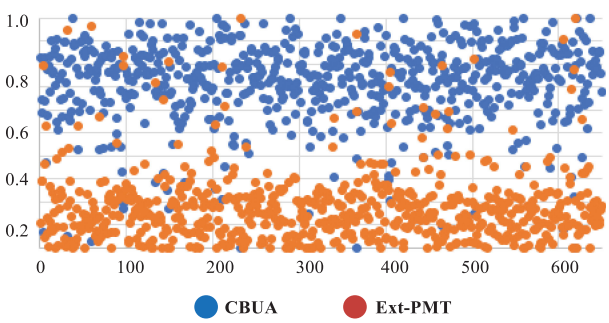
Fig. 13. The distribution of trustworthiness of mutant predictability on 163 O-PMT projects. Each column represents a project. The *i*-th cell in each column (from bottom to top) represents the percentage of mutants with trustworthiness between $0.05^{*}(i\text{-}1)$ and $0.05^{*}i$ in the project.

Among the inspected mutants, "Number of detected" is the number of actually alive mutants. For PMT, we will inspect the sorted mutants one by one according to the survival probability predicted by PMT. We fix the "Number of detected" equals the CBUA's "Number of detected". That is to say, the inspecting process will not end until we have found as many alive mutants as CBUA. On termination, the number of inspected mutants is "Number of inspected". This enables us to examine which approach results in a higher effort to find the same number of "covered but alive mutants". In this table, for each row, a higher ER will be shown in blue background. From Table 9, we have the following observations:

- CBUA is more effective than the random model. As can be seen, except the ER of CBUA for the "MathMutator" mutator, all the other ER values are positive. This means that, overall, CBUA is superior to the random model in finding "covered but alive mutants".
- CBUA exhibits a better effectiveness than Ext-PMT. Compared with Ext-PMT, CBUA have achieved a higher ER in most cases (for 10 out of 17 mutators). When taking into account all the mutators together, we can see that CBUA has a ER of 0.250, leading to a 32.97 percent improvement over Ext-PMT (0.188).

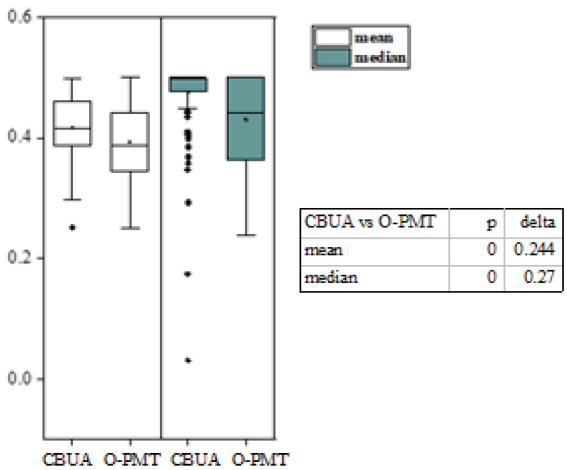


(a)CBUA vs. O-PMT on 163 O-PMT projects



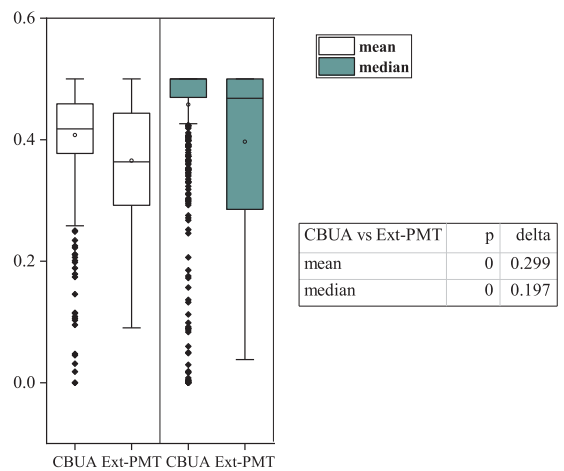
(b) CBUA vs. Ext-PMT on 650 Ext-PMT projects

Fig. 14. Trustworthiness comparison between CBUA and PMT: winning percentage. On each project, there are two points: the blue point indicates the percentage of mutants that CBUA has a higher trustworthiness, while the orange point indicates the percentage of mutants that PMT has a higher trustworthiness.



CBUA vs O-PMT	p	delta
mean	0	0.244
median	0	0.27

(a) CBUA vs. O-PMT on 163 O-PMT projects



CBUA vs Ext-PMT	p	delta
mean	0	0.299
median	0	0.197

(b) CBUA vs. Ext-PMT on 650 Ext-PMT projects

Fig. 15. Trustworthiness comparison between CBUA and PMT: mean/median trustworthiness of mutant predictability.

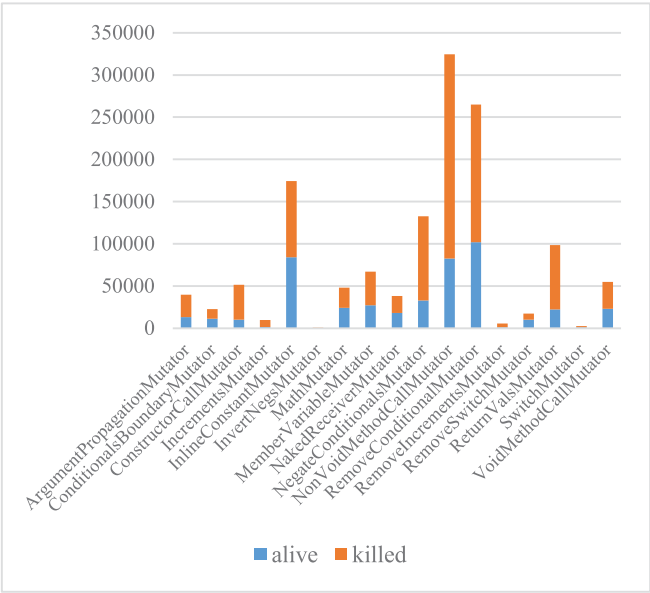


Fig. 16. The distribution of covered mutants over 17 mutators.

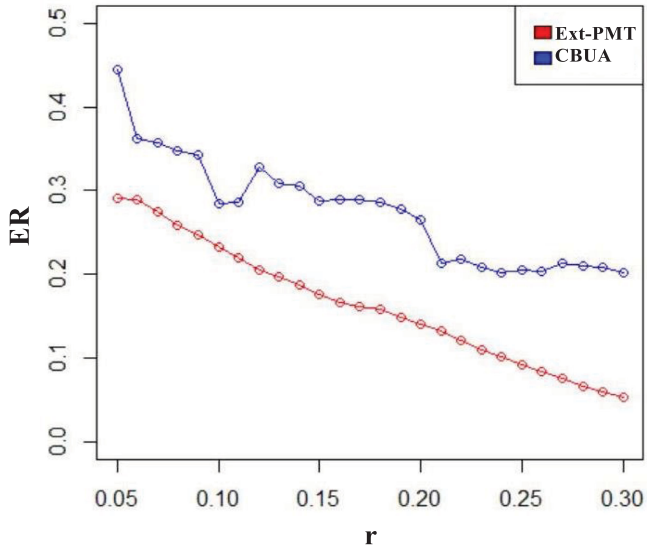


Fig. 17. The overall cost-benefit in finding "covered but alive mutants" by mutant predictability guided method.

Mutant Predictability Guided "Covered But Alive Mutants" Identification. Fig. 17 reports the overall cost-benefit for CBUA and Ext-PMT in finding "covered but alive mutants" under low predictability guided method (the detail corresponding to each mutator is shown in Fig. 18). From Fig. 17 and Fig. 18, we have the following observations:

- CBUA is more effective than the random model. As shown in Fig. 17, when r ranges from 0.05 to 0.30, all the ER values are positive. As shown in Fig. 18, for most mutators, the corresponding ER is much larger than zero. This means that, overall, CBUA is superior to the random model in finding "covered but alive mutants".
- CBUA exhibits a better effectiveness than Ext-PMT. As shown in Fig. 17, for each r, CBUA has a higher ER than Ext-PMT. For most r values, CBUA leads to an improvement in ER is more than 30 percent compared with Ext-PMT. As shown in Fig. 18, in most cases, CBUA has an ER value larger than Ext-PMT. Therefore, overall, CBUA has a higher cost-benefit than Ext-PMT in finding "covered but alive mutants".

Combining the above results, we can see that CBUA is superior to Ext-PMT in helping developers find "covered but alive mutants".

6 Discussion

In this section, we discuss our experimental results. First, we analyze the reason that CBUA performs well. Second, we illustrate when supervised approaches have a potential risk in violating mutation score monotonicity property. Third, we report the influence of test strength on our finding. Fourth, we analyze the additional practical benefits of

TABLE 9
The Cost-Benefit in Finding "Covered But Alive Mutants" by "Alive" Mutant Prediction Guided Method

Mutator	Number of mutants	Number of alive mutants	CBUA			Ext-PMT		
			Number of inspected	Number of detected	ER	Number of inspected	Number of detected	ER
ConstructorCallMutator	51511	10036	10401	4350	0.534	17103	4350	0.234
InvertNegsMutator	392	97	66	37	0.559	110	37	0.264
RemoveIncrementsMutator	5616	820	553	173	0.533	834	173	0.296
ReturnValsMutator	98299	22199	20975	7958	0.405	27594	7958	0.217
IncrementsMutator	9557	1272	1294	387	0.555	1808	387	0.378
NonVoidMethodCallMutator	324551	82415	63476	26073	0.382	81053	26073	0.211
ArgumentPropagationMutator	39547	13230	7578	3872	0.345	8134	3872	0.297
RemoveConditionalMutator	264821	102017	48939	21841	0.137	50796	21841	0.104
RemoveSwitchMutator	17175	9996	2848	2108	0.214	2938	2108	0.189
MemberVariableMutator	66858	27272	15903	8187	0.208	16212	8187	0.192
ConditionalsBoundaryMutator	22506	11107	3550	2282	0.232	3399	2282	0.265
InlineConstantMutator	174465	84152	37337	21520	0.163	35364	21520	0.207
VoidMethodCallMutator	54851	22794	9837	5191	0.213	8711	5191	0.303
NegateConditionalsMutator	132693	33001	24520	8139	0.251	18951	8139	0.421
NakedReceiverMutator	38151	17938	8644	4874	0.166	6851	4874	0.339
SwitchMutator	2377	792	410	165	0.172	256	165	0.483
MathMutator	47991	23997	5009	2211	-0.133	2569	2211	0.419
Total	1351361	463135	261340	119368	0.250	282683	119368	0.188

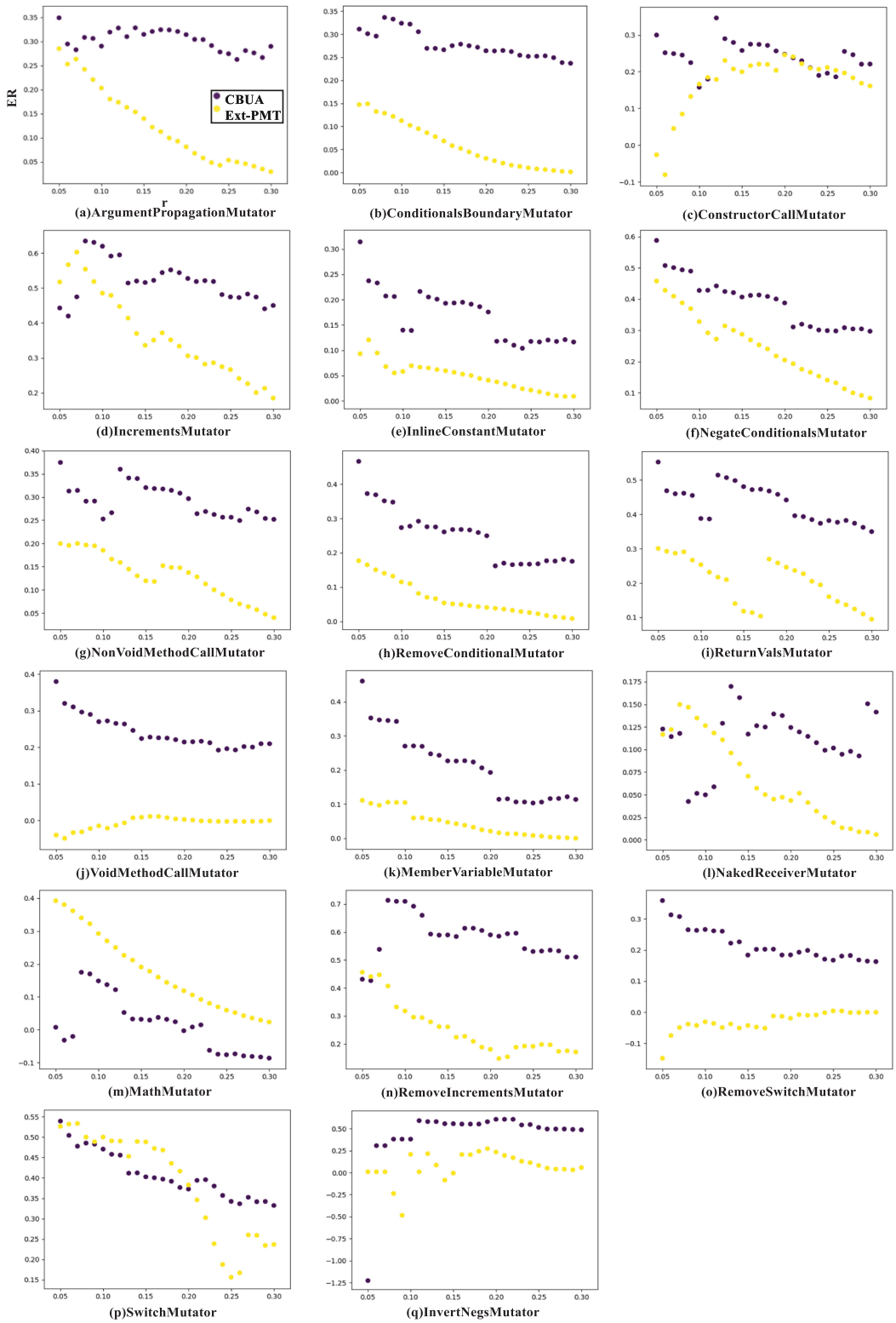
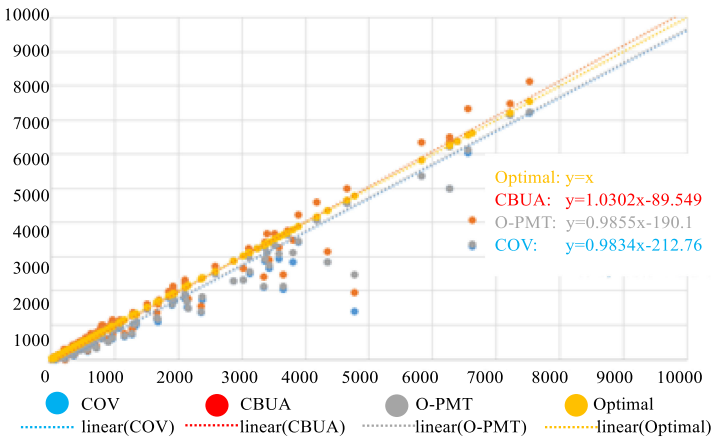


Fig. 18. The cost-benefit in finding "covered but alive mutants" corresponding to each mutator by mutant predictability guided method.

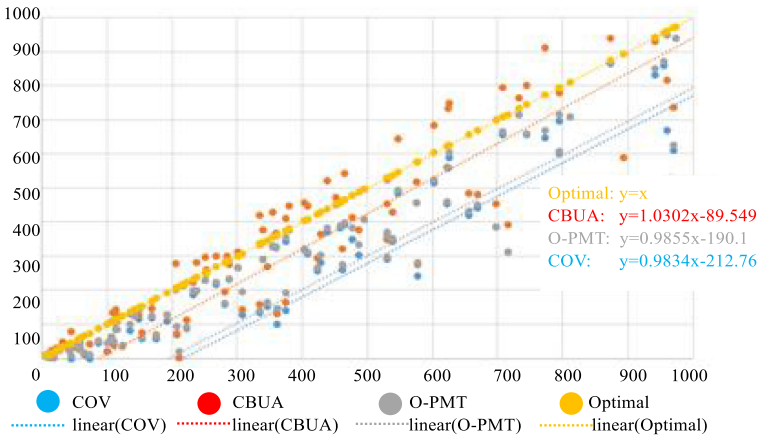
using CBUA over the other approaches. At last, we explain why there is an inconsistent observation on the APE comparison and F_1/AUC comparison.

6.1 Why does CBUA Perform Well in Test Suite Effectiveness Evaluation?

In the following, we use the 163 O-PMT data sets as an example to explain why CBUA has a competitive performance. Fig. 19 shows a scatter plot for reporting the relationship between the number of actual alive mutants (x axis) and the predicted alive mutants (y axis). In this plot, each dot corresponds to a project: its abscissa represents the actual number of alive mutants, while its ordinate represents the predicted number of alive mutants. As can be seen, the color of a dot can be yellow, red, blue, and gray, which respectively indicate the result under the optimal approach (i.e., the number



(a) *x* axis: range from 0 to 10000



(b) *x* axis: range from 0 to 1000

Fig. 19. The scatter plot of the number of alive mutants (*x* axis) against the number of the predicted alive mutants (*y* axis).

of predicted alive mutants is equal to the number of actual alive mutants), CBUA, COV, and O-PMT (the results are from LOO prediction among 163 projects). In order to better understand the performance of each approach, we fit a regression line for the dots under each approach. Fig. 19a is the scatter plot with x axis ranging from 0 to 10000, while Fig. 19b is the scatter plot with x axis ranging from 0 to 1000.

In Fig. 19, the yellow line ($y = x$) is the optimal prediction line. The closer to the yellow line, the better the prediction performance. All the CBUA, O-PMT, and COV regression lines have a R^2 larger than 0.99, indicating a strong linear relationship between the number of actual alive mutants and the number of predicted alive mutants. As can be seen, the CBUA regression line is closer to the optimal prediction line compared with the O-PMT and COV regression lines. In particular, we make the following interesting observation: under CBUA, the number of predicted alive mutants can be larger or smaller than the number of actual alive mutants; however, under both O-PMT and COV, the numbers of predicted alive mutants are almost always smaller than the number of actual alive mutants.

By the above analysis, we know that O-PMT and COV tend to underestimate the number of actual alive mutants. By using E_{alive}, CBUA provides a more accurate estimate of the number of actual alive mutants. This is an intuitive explanation why CBUA works well in test suite effectiveness evaluation compared with O-PMT and COV.

6.2 When Will Supervised Approaches Have a Risk in Violating Mutation Score Monotonicity?

In PMT, a large number of features are used to build a supervised model. Of these features, the following four features are related to the number of test cases in a test suite: *numTestCovered*, *numExecuteCovered*, *numMutantAssertion*, and *numClassAssertion*. For a mutant, the values corresponding to these four features will remain no change or increase when a new test case is added into the test suite. In the following, these four features will be called size-related features.

In a supervised approach, a model is trained to relate features with the mutant label ("killed" or "alive"). For each mutant, the model will produce a predicted probability of being "killed". Intuitively, each size-related feature is positively related with the predicted probability of being "killed", i.e., a larger value indicates a higher predicted probability of being "killed". However, in a multivariate model, a size-related feature may be negatively related with the predicted probability of being "killed" due to suppressor relationships between the features [43]. In an extreme case, if all the size-related features are negatively related with the predicted probability of being "killed", adding a test case into the test suite will result in a decrease in the

TABLE 10
The Mean Value of APE, F_1, AUC, and ER for 141 Large-Scale Projects Under Different Groups

	APE			F_1			AUC			ER		
Project	CBUA	Ext-PMT	COV	CBUA	Ext-PMT	COV	CBUA	Ext-PMT	COV	CBUA	Ext-PMT	COV
low score	0.055	○ 0.119	○ 0.157	0.960	○ 0.811	○ 0.807	0.970	0.929	○ 0.862	0.346	○ 0.175	N/A
moderate score	0.083	0.094	○ 0.162	0.892	0.822	○ 0799	0.937	0.901	○ 0.860	0.265	0.230	N/A
high score	0.107	0.075	○ 0.147	0.732	0.776	0.740	0.834	0.887	0.829	0.229	0.219	N/A
avg.	0.082	0.096	0.156	0.858	0.803	0.782	0.902	0.908	0.850	0.280	0.208	N/A

predicted probability of being "killed". As a result, the predicted mutation score may decrease. Therefore, a supervised approach may have a risk in violating mutation score monotonicity.

In the following, for the simplicity of presentation, we use logistic regression as the representative to expose the fact that supervised approaches may violate mutation score monotonicity (indeed, logistic regression is one of the supervised modeling techniques applicable to O-PMT [13]). Based on the O-PMT project *jieba-analysis*, we train a logistic regression model with the following formula:

$$p = \frac{1}{1 + e^{-f(x)}}. \tag{8}$$

Here,

$$f(x) = w_0 x_0 + w_1 x_1 + \ldots + w_{14} x_{14} + b, \tag{9}$$

$f(x)$ is a linear transformation of x, w_i is the coefficient ($0 \leq i \leq 14$), and p is the predicted probability of being "killed". Consequently, we have:

$$\begin{aligned} f(x) = {} & -0.0005 * numTestCover \\ & + (3.97E-8) * numExecuteCover \\ & - 0 * numMutantAssertion \\ & - 0 * numClassAssertion + \ldots + w_{14} x_{14} + b. \end{aligned} \tag{10}$$

As can be seen, the coefficients of the size-related features are either negative or very close to zero. This means that a decrease in p as the size of the test suite increases. As a result, the number of the predicted "killed" mutants may decrease. Consequently, the predicted mutation score may decrease with the increase of the size of the test suite, i.e., the mutation score monotonicity property may be violated. It should be pointed out that although the above analysis is based on logistic regression, it is an inherent problem to supervised modeling techniques, not unique to logistic regression.

6.3 What is the Influence of Test Strength on Our Finding?

In our experiment, the projects under study may undergo different test strength. As a result, test strength may have a confounding effect on our finding. Previous studies have shown that, in spite of not being perfect, mutation score can be considered as a proxy measure of test strength [6], [9], [10], [11], [28], [46]. In order to investigate the influence of test strength, we applied the following preprocessing to 650 Ext-PMT data sets. First, we filtered out all projects having mutants less than 5000. The purpose of this was to exclude the influence of small projects on our finding. As a result, 141 projects were remained. Second, we divided 141 projects into three groups with the same size according to their actual mutation scores: low score group, moderate score group, and high score group. Consequently, we obtained 47 projects in low score group (mutation score: (0, 0.055]), 47 projects in moderate score group (mutation score: (0.055, 0.365]), and 47 projects in high score group (mutation score: (0.365, 0.838]). In nature, we use the distribution of mutation score to obtain these groups, similar to the strategies taken in the literature [10], [11], [12].

Table 10 summarizes the experimental results on the 141 projects. In Table 10, the first column shows group name. The second to fifth columns, respectively, show the APE, F_1, AUC, and ER of three approaches. For each group, this table not only reports the mean APE, F1, and AUC but also reports the ER based on the "alive" mutant prediction guided method. In particular, a cell shown in red background means that the corresponding approach is significantly worse than CBUA (Benjamini-Hochberg corrected p-value < 0.05), while a cell shown in blue background indicates that the corresponding approach is significantly better (Benjamini-Hochberg corrected p-value < 0.05). Furthermore, a circle indicates a nontrivial effect size (Cliff's $\delta > 0.147$), i.e., there is a practically important difference.

If we ignore the statistical significance and the effect size, we have the following interesting observation. When compared with Ext-PMT, in terms of APE, F_1, and AUC, CBUA is better in the low and moderate score groups but is worse in the high score group. In terms of ER, CBUA is better, regardless of which group is considered (note that their difference decreases with mutation score). As a result, it seems that, in the early and middle stages of testing, PMT is not necessary at all while CBUA can provide simple and practical results. When developers have a test suite with powerful strength and want to further strengthen it, it may be worth using PMT. This is especially true when the objective is to obtain accurate mutation score prediction or mutant label prediction.

If we take into account the statistical significance and the effect size, we have the following observations:

- Mutation score prediction. Compared with COV, CBUA is significantly and practically better, regardless of which group is taken into account. Compared with Ext-PMT, COV is superior in the low score group and is similar in the moderate and high score groups.
- Mutant label prediction. Compared with COV, CBUA is superior in the low and moderate score groups and is similar in the high score group. Compared with Ext-PMT, CBUA is similar or even better in the low and moderate score groups, regardless of whether F_1 or AUC is considered. For the high score

group, CBUA has a similar F_1 and an inferior AUC compared with Ext-PMT.
- "Covered but alive mutants" identification. Since COV is not applicable in this scenario, the corresponding column has a value of "N/A". Compared with Ext-PMT, CBUA has a competitive ER, regardless of which group is considered. In the low score group, the advantage of CBUA in ER is practically important.

In summary, CBUA has a very apparent advantage over COV and is very competitive to Ext-PMT, regardless of which group is considered. This means that test strength does not have a large influence on our conclusion that CBUA is a simple yet effective approach. Anyway, it is safe to conclude that CBUA is worthy of being proposed as a baseline approach for predictive mutation testing.

6.4 What are the Practical Benefits of Using CBUA Over the Other Approaches in Addition to "Covered But Alive Mutants" Identification?

The biggest advantage of using CBUA instead of supervised approaches is that CBUA does not have any more overhead than simple statement coverage testing. For CBUA, only *numTestCovered*, the number of test cases covering the mutated statement, is needed to be collected. However, for supervised approaches such as PMT, a large number of features are needed to be collected. In practice, it may be challenging to collect such features. For example, in [13], Zhang *et al.* reported that, of the 388 single-packaged projects that were successfully built with Maven and passed all their Junit tests, 234 projects cannot be handled by their feature collecting tools (including PIT). In [44], Mao *et al.* reported that 2299 out of 2953 projects faced the same problem. In this sense, it may be difficult to deploy such an approach to a large company.

When compared with COV, CBUA exhibits a superior accuracy in mutation score prediction and is able to provide mutant predictability. Given this fact, why not use a more effective prediction approach with the same cost?

6.5 Why There is an Inconsistent Observation on the APE Comparison and F_1/AUC Comparison?

We find that, compared with PMT, CBUA tends to have a better APE but a worse F_1/AUC. In other words, CBUA tends to perform better in mutation score prediction but worse in mutant label prediction. The inconsistency is mainly caused by the different number of features used in CBUA and PMT. As mentioned before, CBUA uses only one feature (i.e., *numTestCovered*) to classify mutants into "alive" or "killed". If a number of mutants have the same value of *numTestCovered*, they will have the same predicted label. Unlike CBUA, PMT uses many more features to classify mutants into "alive" or "killed". As a result, under PMT, these mutants having the same *numTestCovered* can be classified into different groups (i.e., "alive" or "killed"). In other words, with the use of many more features, PMT may have a stronger ability to distinguish between alive and killed mutants. Nonetheless, our experimental results in Section 5 show that PMT only leads to a limited improvement in F_1/AUC compared with CBUA (in most cases, a trivial or near-trivial improvement).

7 Threats to Validity

In this section, we discuss the threats to the construct validity, internal validity, and external validity of our study.

7.1 Construct Validity

Construct validity denotes the extent to which the variables used in our study accurately measure what they purport to measure. We use the O-PMT data sets shared by Zhang *et al.*'s study [13] to conduct the experiments, thus enabling a direct comparison of our proposed CBUA and their O-PMT. According to Zhang *et al.*'s study, they employed various widely used libraries and frameworks to collect the independent and dependent variables in order to ensure the construct validity. Nonetheless, we did find that there were noise instances in their data sets. In our study, in order to reduce the threats to construct validity, we filtered out noise instances before conducting the data analyses. At the same time, we use the 650 Ext-PMT data sets, which have no noise instances, shared by Mao *et al.*'s study [44] to conduct the experiments. In this sense, the threat to construct validity in our study has been minimized.

7.2 Internal Validity

Internal validity denotes the extent to which a causal conclusion based on a study is warranted. In our study, we built an unsupervised model to predict test suite effectiveness. The main threat comes from the existence of the causal relationship between independent variables (i.e., *numTestCovered*) and the dependent variable (i.e., whether a test is killed or alive). From our observations and previous work, we find that there is a correlation between them. In particular, we use an approximate approach rather than an accurate analysis to estimate the survival probability of each mutant. To reduce the threats to internal validity, we constrain CBUA with the score monotonicity property. Nonetheless, these problems need to be further investigated in the future work.

7.3 External Validity

The most important threat to the external validity is that our findings may not be generalized to other projects and settings. In our experiments, we use a large number of projects, which are across a wide range of project types and scales, as the target projects. In particular, we use different mutation testing tools to generate mutants, use different ways to generate test suites, and take into account high-order mutants. The experimental results drawn from these projects and settings are quite consistent. Nonetheless, since the subject target projects under study might not be representative of projects in general, we do not claim that our findings can be generalized to all projects and settings. Indeed, this is an inherent problem to most (if not all) empirical studies, not unique to us. To mitigate this threat, there is a need to replicate our study across a wider variety of projects and settings in the future.

8 Conclusions and Future Work

In this paper, we propose a Coverage-Based Unsupervised Approach (CBUA) to evaluate the effectiveness of a test

suite. As an unsupervised method, CBUA only requires a one-time execution of the test suite against the target production code, without requiring any mutant execution and any training data. Furthermore, CBUA satisfies the score monotonicity property which may be violated by supervised approaches. The experimental results from a large number of projects show that CBUA has a promising prediction performance on both mutation score prediction and mutant label prediction. When predicting mutation scores, CBUA performs at least as well as the state-of-the-art supervised method PMT and significantly better than the simple unsupervised method COV. At the same time, CBUA guarantees a reasonable F_1 and AUC with a higher trustworthy predictability than PMT when classifying a mutant as killed or alive. In particular, CBUA is more suitable than PMT to be applied in the scenario where limited training data are available. What is more important, CBUA shows a competitive cost-benefit in finding "covered but alive mutants" compared with PMT (especially when the mutation testing coverage is low), where COV is not applicable. Considering the advantages of CBUA, we strongly suggest that CBUA should be used as a baseline approach for comparison when any novel approach is proposed to evaluate test suite effectiveness.

In the future work, on the one hand, we plan to use more projects to further investigate the effectiveness of CBUA. On the other hand, we will explore the application of CBUA in many test activities such as selecting fault revealing mutants [54], test suite reduction, test suite augmentation, and test case prioritization.

Acknowledgements

This work is partially supported by the National Key R&D Program of China (2018YFB1003901) and the National Natural Science Foundation of China (61772259, 61872177, 61832009, 61772263).

References

[1] I. Ahmed, R. Gopinath, C. Brindescu, A. Groce, and C. Jensen, "Can testedness be effectively measured," in *Proc. 24th ACM SIGSOFT Int. Symp. Foundations Softw. Eng.*, 2016, pp. 547–558.

[2] X. Cai, "Coverage-based testing strategies and reliability modeling for fault-tolerant software systems," Ph.D. dissertation, Chinese Univ. Hong Kong, Hong Kong, 2006.

[3] X. Cai and M. R. Lyu, "The effect of code coverage on fault detection under different testing profiles," *ACM SIGSOFT Softw. Eng. Notes*, vol. 30, no. 4, pp. 1–7, 2005.

[4] P. S. Kochhar, F. Thung, and D. Lo, "Code coverage and test suite effectiveness: Empirical study with real bugs in large systems," in *Proc. IEEE 22nd Int. Conf. Softw. Anal. Evol. Reeng.*, 2015, pp. 560–564.

[5] P. Ammann and J. Offutt. in *Syntax-Based Testing*. 2nd ed, NY, NY, USA: Cambridge Univ. Press, 2016, pp. 320–325.

[6] J. H. Andrews, L. C. Briand, and Y. Labiche, "Is mutation an appropriate tool for testing experiments?," in *Proc. 27th Int. Conf. Softw. Eng.*, 2005, pp. 402–411.

[7] R. A. DeMillo, R. J. Lipton, and F. G. Sayward, "Hints on test data selection: Help for the practicing programmer," *IEEE Comput.*, vol. 11, no. 4, pp. 34–41, Apr. 1978.

[8] R. G. Hamlet, "Testing programs with the aid of a compiler," *IEEE Trans. Softw. Eng.*, vol. 3, no. 4, pp. 279–290, Jul. 1977.

[9] Y. Jia and M. Harman, "An analysis and survey of the development of mutation testing," *IEEE Trans. Softw. Eng.*, vol. 37, no. 5, pp. 649–678, Sep./Oct. 2010.

[10] K. Jalbert and J. S. Bradbury, "Predicting mutation score using source code and test suite metrics," in *Proc. 1st Int. Workshop Realizing AI Synergies Softw. Eng.*, 2012, pp. 42–46.

[11] K. Jalbert, "Predicting mutation score using source code and test suite metrics," M.S. thesis, University of Ontario Institute of Technology, 2012.

[12] G. Grano, F. Palomba, and H.C. Gall, "Lightweight assessment of test-case effectiveness using source-code-quality indicators," *IEEE Trans. Softw. Eng.*, 2019. to be published, doi: 10.1109/TSE.2019.2903057.

[13] J. Zhang, L. Zhang, M. Harman, D. Hao, Y. Jia, and L. Zhang, "Predictive mutation testing," *IEEE Trans. Softw. Eng.*, vol. 45, no. 9, pp. 898–918, Sep. 2019.

[14] I. Zliobaite, M. Pechenizkiy, and J. Gama, "An overview of concept drift applications," in *Big Data Analysis: New Algorithms For a New Society*. Berlin, Germany: Springer, 2015, vol. 16, pp. 91–114.

[15] J. Ekanayake, J. Tappolet, H.C. Gall, and A. Bernstein, "Tracking concept drift of software projects using defect prediction quality," in *Proc. 6th IEEE Int. Working Conf. Mining Softw. Repositories*, 2009, pp. 51–60.

[16] B. Kurtz, P. Ammann, M. E. Delamaro, J. Offutt, and L. Deng, "Mutant subsumption graphs," in *Proc. IEEE 7th Int. Conf. Softw. Test. Verif. Valid. Workshops*, 2014, pp. 176–185.

[17] R. Just, B. Kurtz, and P. Ammann, "Inferring mutant utility from program context," in *Proc. 26th ACM SIGSOFT Int. Symp. Softw. Test. Anal.*, 2017, pp. 284–294.

[18] A. J. Offutt, A. Lee, G. Rothermel, R. H. Untch, and C. Zapf, "An experimental determination of sufficient mutant operators," *ACM Trans. Softw. Eng. Methodol.*, vol. 5, no. 2, 1996, pp. 99–118.

[19] R. J. Lipton and F. G. Sayward, "The status of research on program mutation," in *Proc. Workshop Softw. Test. Test Documentation*, 1978, pp. 355–373.

[20] A. S. Namin, J. H. Andrews, and D. J. Murdoch, "Sufficient mutation operators for measuring test effectiveness,"in *Proc. ACM/IEEE 30th Int. Conf. Softw. Eng.*, 2008, pp. 351–360.

[21] M. Gligoric, A. Groce, C. Zhang, R. Sharma, M. A. Alipour, and D. Marinov, "Guidelines for coverage-based comparisons of non-adequate test suites," *ACM Trans. Softw. Eng. Methodol.*, vol. 24, no. 4, pp. 1–33, 2015.

[22] L. Inozemtseva and R. Holmes, "Coverage is not strongly correlated with test suite effectiveness," in *Proc. 36th Int. Conf. Softw. Eng.*, 2014, pp. 435–445.

[23] Y. Zhang and A. Mesbah, "Assertions are strongly correlated with test suite effectiveness," in *Proc. 10th Joint Meeting Foundations Softw. Eng.*, 2015, pp. 214–224.

[24] A. S. Namin and J. H. Andrews, "The influence of size and coverage on test suite effectiveness," in *Proc. 18th Int. Symp. Softw. Test. Anal.*, 2009, pp. 57–68.

[25] J. H. Andrews, L. C. Briand, Y. Labiche, and A. S. Namin, "Using mutation analysis for assessing and comparing testing coverage criteria," *IEEE Trans. Softw. Eng.*, vol. 32, no. 8, pp. 608–624, Aug. 2006.

[26] P. G. Frankl, S. N. Weiss, and C. Hu, "All-uses vs mutation testing: An experimental comparison of effectiveness," *Elsevier J. Syst. Softw.*, vol. 38, no. 3, pp. 235–253, 1997.

[27] A. Zeller, R. Gopinath, M. Böhme, G. Fraser, and C. Holler, "Generating software tests: Breaking software for fun and profit," 2019. [Online]. Available: https://www.fuzzingbook.org/

[28] R. Just, D. Jalali, L. Inozemtseva, M.D. Ernst, R. Holmes, and G. Fraser, "Are mutants a valid substitute for real faults in software testing?," in *Proc. 22nd ACM SIGSOFT Int. Symp. Foundations Softw. Eng.*, 2014, pp. 654–665.

[29] J. H. Andrews, L. C. Briand, and Y. Labiche, "Is mutation an appropriate tool for testing experiments?," in *Proc. 27th Int. Conf. Softw. Eng.*, 2005, pp. 402–411.

[30] H. Watanabe, "Effectiveness of inadequate test suites, a case study of mutant analysis," M.S. thesis, KTH Royal Institute of Technology, Sweden, 2017.

[31] S. J. Russell and P. Norvig, in *Artificial Intelligence: A Modern Approach*. Pearson, USA, 3rd ed., 2010.

[32] Y. Zhou *et al.*, "How far we have progressed in the journey? An examination of cross-project defect prediction," *ACM Trans. Softw. Eng. Methodol.*, vol. 27, no. 1, pp. 1–51, 2018.

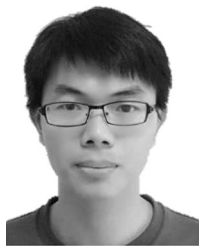
[33] K. N. King and A. J. Offutt, "A Fortran language system for mutation-based software testing," *Wiley Softw.: Practice Experience*, vol. 21, no. 7, pp. 685–718, 1991.

[34] D. Schuler and A. Zeller, "Javalanche: Efficient mutation testing for java," in *Proc. FSE*, 2009, pp. 297–298.
[35] L. Zhang, D. Marinov, L. Zhang, and S. Khurshid, "Regression mutation testing," in *Proc. Int. Symp. Softw. Test. Anal.*, 2012, pp. 331–341.
[36] D. Tengeri *et al.*, "Relating code coverage, mutation score and test suite reducibility to defect density," in *Proc. IEEE 9th Int. Conf. Softw. Test. Verif. Valid. Workshops*, 2016, pp. 174–179.
[37] PIT, 2019. [Online]. Available: http://pitest.org/
[38] R. Just, F. Schweiggert, and G. M. Kapfhammer, "Major: An efficient and extensible tool for mutation analysis in a java compiler," in *Proc. 26th IEEE/ACM Int. Conf. Autom. Softw. Eng.*, 2016, pp. 612–615.
[39] randoop, 2019. [Online]. Available: https://randoop.github.io
[40] D. Rey and M. Neuhäuser. in *Wilcoxon-Signed-Rank Test*. Berlin Germany: Springer, 2011.
[41] N. Cliff, "Dominance statistics: Ordinal analyses to answer ordinal questions," *APA Psychol. Bull.*, vol. 114, no. 3, pp. 494–509, 1993.
[42] J. Cohen. in *Statistical Power Analysis For the Behavioral Sciences*. Second Ed. Hillsdale, NJ, USA: Lawrence Erlbaum Associatesm, 2008.
[43] L. C. Briand, J. Wüst, J. W. Daly, and D. V. Porter, "Exploring the relationships between design measures and software quality in object-oriented systems," *J. Syst. Softw.*, vol. 51, no. 3, pp. 245–273, 2000.
[44] D. Mao, L. Chen, and L. Zhang, "An extensive study on cross-project predictive mutation testing," in *Proc. ICST*, 2019, pp. 160–171.
[45] T. Laurent, and A. Ventresque, "PIT-HOM: An extension of pitest for higher order mutation analysis," in *Proc. IEEE Int. Conf. Softw. Test. Verif. Valid. Workshops*, 2019, pp. 83–89.
[46] M. Papadakis, M. Kintis, J. Zhang, Y. Jia, Y. Traona, and M. Harmanb, "Mutation testing advances: An analysis and survey," *Adv. Comput.*, vol. 112, pp. 275–378, 2019.
[47] Y. Zhou, B. Xu, H. Leung, and L. Chen, "An in-depth study of the potentially confounding effect of class size in fault prediction," *ACM Trans. Softw. Eng. Methodol.*, vol. 23, no. 1, pp. 1–51, 2014.
[48] T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell, "A systematic literature review on fault prediction performance in software engineering," *IEEE Trans. Softw. Eng.*, vol. 38, no. 6, pp. 1276–1304, Nov./Dec. 2012.
[49] [Online]. Available: https://github.com/cobertura/cobertura/issues/337
[50] Y. Yang *et al.*, "Hunting for bugs in code coverage tools via randomized differential testing," in *Proc. ICSE*, 2019, pp. 488–498.
[51] Y. Yang *et al.*, "Automatic self-validation for code coverage profilers," in *Proc. 34th IEEE/ACM Int. Conf. Autom. Softw. Eng.*, 2019, pp. 79–90.
[52] A. Arcuri and L.C. Briand, "A practical guide for using statistical tests to assess randomized algorithms in software engineering," in *Proc. 33rd Int. Conf. Softw. Eng.*, 2011, pp. 1–10.
[53] I. Rodriguez, L. Llana, and P. Rabanal, "A general testability theory: Classes, properties, complexity, and testing reductions," *IEEE Trans. Softw. Eng.*, vol. 40, no. 9, pp. 862–894, Sep. 2014.
[54] T. Chekam, M Papadaki, T. Bissyandé, Y. Traon, and K. Sen, "Selecting fault revealing mutants," *Empir. Softw. Eng.*, vol. 25, no. 1, pp. 434–487, 2020.

Peng Zhang is currently working toward the PhD degree in the Department of Computer Science and Technology at Nanjing University. His current research direction is the evaluation and application of test suite effectiveness.

Yanhui Li is a researcher with the Department of Computer Science and Technology at Nanjing University. His current research direction is empirical software engineering.

Wanwangying Ma is currently working toward the PhD degree in the Department of Computer Science and Technology at Nanjing University. Her research interests include program analysis and software metrics.



Yibiao Yang received the BS degree from Southwest Jiaotong University, in 2010. He is currently a researcher with the Department of Computer Science and Technology at Nanjing University. His main research interests include empirical software engineering and software analysis.

Lin Chen received the PhD degree in computer science from Southeast University, in 2009. He is currently an associate professor with the Department of Computer Science and Technology at Nanjing University. His research interests are software analysis and software maintenance.

Hongmin Lu received the PhD degree in computer science from Southeast University, in 2013. She is a researcher with the Department of Computer Science and Technology at Nanjing University. Her research interests are software metrics and software maintenance.

Yuming Zhou is a professor with the Department of Computer Science and Technology at Nanjing University. His main research interests are empirical software engineering.

Baowen Xu (Member, IEEE) received the BS degree from Wuhan University, the MS degree from the Huazhong University of Science and Technology, and the PhD degree from Beihang University, all in computer science. He is currently a professor with the Department of Computer Science and Technology at Nanjing University. His main research interests are programming languages, software testing, software maintenance, and software metrics. He is a member of the IEEE Computer Society.