Adaptive Metamorphic Testing

1st Chang-ai Sun, Hepeng Dai
School of Computer and Communication Engineering
University of Science and Technology Beijing
Beijing, China
casun@ustb.edu.cn, daihepeng@xs.ustb.edu.cn

2rd Tsong Yueh Chen
Faculty of Information and Communication Technologies
Swinburne University of Technology Australia
Melbourne, Australia
tychen@swin.edu.au

***Abstract*—Metamorphic testing (MT) is a promising technique to alleviate the oracle problem, which fist defines metamorphic relations (MRs) to generate new test cases (i.e. follow-up test cases) from the original test cases (i.e. source test cases), then the results of source and follow-up test cases are verified against the relevant MRs. To improve the effectiveness of MT, researchers have focused their effort on generating or selecting better MRs, while ignoring the impact of test execution. Most of MT techniques employ random testing strategy (RT) to select source test cases. It may not be efficient because RT does not use any information of the software under test. This study aims to improve the efficiency of MT through controlling the execution process of MT, and proposes a adaptive metamorphic testing (AMT) technique. We conduct a empirical study where AMT is employed to test three real-life programs. The results of empirical study show that AMT outperforms traditional MT in terms of fault-detecting efficiency.**

***Index Terms*—metamorphic testing, control test process, random testing**

I. Introduction

Test result verification is an important part of software testing. A test oracle [1] is a mechanism that can exactly decide whether the output produced by a programs is correct. However, there are situations where it is difficult to decide whether the result of the software under test (SUT) agrees with the expected result. This situation is known as oracle problem [2], [3]. In order to alleviate the test oracle, several techniques have been proposed such as N-version testing [4], metamorphic testing (MT) [5], [6], assertions [7], machine learning [8], etc. Among of them, MT obtains metamorphic relations (MRs) according to the properties of SUT. Then, MRs are used to generate new test cases called follow-up test cases from original test cases known as source test cases. Next, both source and follow-up test cases are executed and their result are verified against the corresponding MRs.

The fault-detecting effectiveness of MT relies on the quality of MRs and the source test cases. There are astronomically large number of studies to investigate generate good MRs [9]–[14] or the source test cases [11], [15], [16]. However, Researchers ignore the impact of test execution on the efficiency of MT. Random testing (RT) that randomly selects test cases from input domain (which refers to the set of all possible inputs of SUT), which is most commonly used technique in traditional MT [17]. Although RT is simple to implement, RT does not make use of any execution information about SUT or the test history. Thus, traditional MT may be inefficient in some situations.

In contrast to RT, partition testing (PT) attempts to generate test cases in a more systematic way, aiming to use fewer test cases to reveal more faults. When conducting PT, the input domain of SUT is divided into disjoint partitions, with test cases then selected from each and every one. Each partition is expected to have a certain degree of homogeneity, that is, test cases in the same partition should have similar software execution behavior: If one input can detect a fault, then all other inputs in the same partition should also be able to detect a fault. RT and PT have their own characteristics. Therefore, it is a nature thought to investigate the integration of them for developing new testing techniques. Sun et al. [18] proposed adaptive partition testing (APT) that takes advantages of testing information to control the testing process, with the goal of benefitting from the advantages of RT and PT.

In this study, we investigate how to make use of feedback information in the previous tests to control the execution process of MT, and propose a adaptive metamorphic testing framework based on PT to improve the fault-detecting efficiency of MT. We conduct an empirical study to evaluate proposed technique. The paper makes the following contributions:

1) An adaptive metamorphic testing framework is proposed, which combines the basic principle of MT and software cybernetics [19].
2) An MRs-centric adaptive metamorphic testing technique (M-AMT) which used a feedback mechanism to control the execution process of MT.
3) An empirical study has been conducted to evaluate the fault detection efficiency of the proposed AMT, where three Java programs are selected as subjects to compare the performance of traditional MT and AMT in terms of fault detection efficiency and time overhead.

The rest of the paper is organized as follows. Section II introduces the underlying concepts related to MT and DRT. Section III presents the motivation of this study, the framework and algorithms of AMT. Section IV describes an empirical study where the proposed AMT is used to test three real-life programs, the results of which are summarized in Section V. The related work is reviewed in Section VI. Finally, we summarize our work in Section VII.

II. Background

In this section, we present the basics to understand our approach. we start with a brief introduction to metamorphic testing, and then describe dynamic random testing.

A. Metamorphic Testing

Let us use a simple example to illustrate how MT works. For instance, consider the mathematic function $f(x, y)$ that can calculate the maximal value of two integers x and y. There is a simple yet obvious property: the order of two parameters x and y does not affect the output, which can be described as the follow metamorphic relation (MR): $f(x, y) = f(y, x)$. In this MR, (x, y) is source test case, and (y, x) is considered as follow-up test case. Suppose `P` denotes a program that implements the function $f(x, y)$, `P` is executed with a test cases $(1, 2)$ and $(2, 1)$. Then we check $P(1, 2) = P(2, 1)$: If the equality does not hold, then we consider that `P` at least has one fault.

B. Dynamic Random Testing

Cai et al [20] proposed dynamic random testing (DRT), which adjusts the test profile according to the result of current test. In DRT, the input domain is first divided into m partitions (denoted $s_1, s_2, \ldots, s_m$), and each s_i is assigned a probability p_i. During the test process, the selection probabilities of partitions are dynamically updated. Suppose that a test case tc from s_i $(i = 1, 2, \ldots, m)$ is selected and executed. The process of DRT adjusting the value of p_i is as follows: If tc detects a fault, $\forall j = 1, 2, \ldots, m$ and $j \neq i$, we then set

$$p'_j = \begin{cases} p_j - \dfrac{\epsilon}{m-1} & \text{if } p_j \geq \dfrac{\epsilon}{m-1} \\ 0 & \text{if } p_j < \dfrac{\epsilon}{m-1} \end{cases}, \tag{1}$$

where ϵ is a probability adjusting factor, and then

$$p'_i = 1 - \sum_{\substack{j=1 \\ j \neq i}}^{m} p'_j. \tag{2}$$

Alternatively, if tc does not detect a fault, we set

$$p'_i = \begin{cases} p_i - \epsilon & \text{if } p_i \geq \epsilon \\ 0 & \text{if } p_i < \epsilon \end{cases}, \tag{3}$$

and then for $\forall j = 1, 2, \ldots, m$ and $j \neq i$, we set

$$p'_j = \begin{cases} p_j + \dfrac{\epsilon}{m-1} & \text{if } p_i \geq \epsilon \\ p_j + \dfrac{p'_i}{m-1} & \text{if } p_i < \epsilon \end{cases}. \tag{4}$$

The detailed DRT algorithm is given in Algorithm 1. In DRT, a test case is selected from a partition that has been randomly selected according to the test profile $\{\langle s_1, p_1\rangle, \langle s_2, p_2\rangle, \ldots, \langle s_m, p_m\rangle\}$ (Lines 2 to 4). If a fault is detected, then Formulas 1 and 2 are used to adjust the values of p_i (Line 6), otherwise Formulas 3 and 4 are used (Line 8). This process is repeated until a termination condition is satisfied (Line 1). Examples of termination conditions can be "when the testing resources are exhausted," "when a certain number of test cases have been executed," "when the first faults is detected," etc.

In AMT, When a source test case and corresponding follow-up test case belong to same partition s_i, DRT is employed to update the test profile: If a fault is detected (i.e., their results violate the related MR), then Formulas 1 and 2 are used to adjust the values of p_i, otherwise Formulas 3 and 4 are used. During the testing process, there is the other situation where a source test case and corresponding follow-up test case do not belong to same partition. In order to adjust the test profile in this situation, we proposed a new strategy that is described in Section III-C.

Algorithm 1 DRT

Input: $\epsilon, p_1, p_2, \ldots, p_m$

1: **while** termination condition is not satisfied **do**
2: Select a partition s_i according to the test profile $\{\langle s_1, p_1\rangle, \langle s_2, p_2\rangle, \ldots, \langle s_m, p_m\rangle\}$.
3: Select a test case tc from s_i.
4: Test SUT using tc.
5: **if** a fault is detected by tc **then**
6: Update test profile according to Formulas 1 and 2.
7: **else**
8: Update test profile according to Formulas 3 and 4.
9: **end_if**
10: **end_while**

III. Adaptive Metamorphic Testing

In this section, we first describe the motivation, then present a framework for improving the efficiency of MT, and finally present an algorithm for AMT, namely MRs-centric adaptive metamorphic testing (M-AMT).

A. Motivation

Since MT was first published, a considerable number of studies have been reported from various aspects [17]. To improve the effectiveness of MT, most of studies have paid their attention to identify the better MRs, which are more likely to be violated. For the effectiveness of MRs, several factors such as the difference between the source and follow-up test cases [9], [10] and the the detecting-faults capacity of MRs compared to existing test oracles [21], have been investigated.

Since the follow-up test cases are generated based on source test cases and MRs, in addition to the so-called good MRs, source test cases also have an impact on the efficiency of MT. However, 57% of existing studies employed RT to select test cases, and 34% of existing studies used existing test suites according to a survey report by Segura et al. in their Journal [17]. In this study, we investigate the use of feedback information to select the next MR and source test case, and its impacts on the fault-detecting efficiency of MT.

This study combines RT and PT to benefit from the advantages of both. During the testing, we make using of feedback information to update the test profile, and a new source test case is selected from a partition that was randomly selected according to updated test profile. Meanwhile, an MR is randomly selected from the set of MRs whose source test cases belong to selected partition.

B. Framework

To show the idea of controlling the execution precess of MT, we propose a AMT framework, as illustrated in Figure 1. In the figure, interactions between MT components and the PT components are depicted in the framework. We next discuss the individual framework components.

1 *Partition Construction.* Partition testing (PT) refers to a class of testing techniques that break the input domain into a number of partitions [22]. Various approaches and principles for achieving convenient and effective partitions have been discussed in the literature [22]–[25]. The input domain of the SUT can be partitioned based on the SUT specifications. Once partitioned, testers can assign probability distributions to the partitions as an initial test profile. This initial test profile can be assigned in different ways, including using a uniform probability distribution, or one that sets probabilities according to the importance of the partition. For instance, a partition should be given a higher priority if more faults have been detected within it in the previous testing history.

2 *Partition Selection* randomly selects a partition according to the test profile.

3 *MR and Source Test Case Selection.* This component is responsible to select an metamorphic relation MR_i based on some strategies from the set of MRs whose source test cases belong to selected partition, then randomly select a source test case stc from the selected partition.

4 *Follow-up Test Case Generation.* This component derives the follow-up test cases ftc by transforming the source test case stc according to the MR_i.

5 *Test Case Execution.* This component executes SUT with source and follow-up test cases, and intercepts their results.

6 *Test Profile Adjustment.* After executing the source and follow-up test cases, its pass or fail status is determined by verifying the results of source and follow-up test cases against the corresponding MR. The pass or fail status is then collected to make adjustments to prepare for the selection of next partition. It has been pointed out that fault-detecting inputs tend to cluster into "continuous regions" [26], [27], that is, the test cases in some partition are more likely to detect faults than the test cases in other partitions. Following the above idea, AMT source test cases and MRs selection are in accordance with the test profile that is dynamically updated according to the feedback information. The strategies updating test profile will be described in Section II-B and III-C.

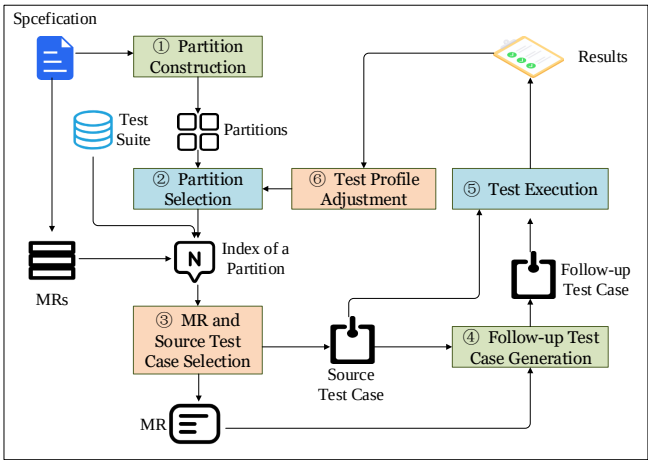

Fig. 1. The framework of adaptive metamorphic testing

C. Metamorphic Dynamic Random Testing

When the source test case stc and follow-up test case ftc related to an metamorphic relation MR belong to the same partition s_i, DRT is suitable to adjust the test profile. However, during the test process, there are some scenarios where the stc is not in the same partition as ftc. In such scenarios, DRT cannot be used to adjust the test profile. To solve this problem, we proposed metamorphic dynamic random testing (MDRT).

Suppose that source test case stc and follow-up test case ftc related to an metamorphic relation MR, belong to s_i and s_f ($f \in \{1, 2, \ldots, m\}, f \neq i$), respectively. If their results violate the MR, $\forall j = 1, 2, \ldots, m$ and $j \neq i, f$, we set

$$p_j' = \begin{cases} p_j - \dfrac{2\epsilon}{m-2} & \text{if } p_j \geq \dfrac{2\epsilon}{m-2} \\ 0 & \text{if } p_j < \dfrac{2\epsilon}{m-2} \end{cases}, \tag{5}$$

where ϵ is a probability adjusting factor, and then

$$p_i' = p_i + \frac{(1 - \sum_{j=1, j\neq i,m}^{m} p_j^{'}) - p_i - p_f}{2}, \tag{6}$$

$$p_f' = p_f + \frac{(1 - \sum_{j=1, j\neq i,m}^{m} p_j^{'}) - p_i - p_f}{2}. \tag{7}$$

Alternatively, if their results hold the MR_h, we set

$$p_i' = \begin{cases} p_i - \epsilon & \text{if } p_i \geq \epsilon \\ 0 & \text{if } p_i < \epsilon \end{cases}, \tag{8}$$

$$p_f' = \begin{cases} p_f - \epsilon & \text{if } p_f \geq \epsilon \\ 0 & \text{if } p_f < \epsilon \end{cases}, \tag{9}$$

and then for $\forall j = 1, 2, \ldots, m$ and $j \neq i, f$, we set

$$p_j' = p_j + \frac{(p_i - p_i^{'}) + (p_f - p_f^{'})}{m-2} \tag{10}$$

The detailed MDRT algorithm is given in Algorithm 2. In MDRT, the source test case is selected from a partition

that has been randomly selected according to the test profile $\{\langle s_1, p_1\rangle, \langle s_2, p_2\rangle, \ldots, \langle s_m, p_m\rangle\}$, and an metamorphic relation is select according to some strategies (Lines 2 to 4 in Algorithm 1). During the testing, if the source and follow-up test cases are not in same partition, then the test profile is updated by changing the p_i: If a fault is detected, then Formulas 5, 6, and 7 are used to adjust the values of p_i (Line 8), otherwise Formulas 8, 9, and 10 are used (Line 10). The testing process is stopped as long as the termination condition is satisfied.

Algorithm 2 MDRT

Input: $\epsilon, p_1, p_2, \ldots, p_m, MR_1, MR_2, \ldots, MR_n$

1: **while** termination condition is not satisfied **do**
2: Select a partition s_i according to the test profile $\{\langle s_1, p_1\rangle, \langle s_2, p_2\rangle, \ldots, \langle s_m, p_m\rangle\}$.
3: Select a source test case stc_i from s_i, and an metamorphic relation MR_h ($h \in \{1, 2, \ldots, n\}$).
4: Based on the MR_h, follow-up test case ftc_i is generated from stc_i, belonging to partition s_f.
5: Test the SUT using stc_i and ftc_i.
6: **if** $i \neq f$ **then**
7: **if** the results of stc_i and ftc_i violate the MR **then**
8: Update $\{\langle s_1, p_1\rangle, \langle s_2, p_2\rangle, \ldots, \langle s_m, p_m\rangle\}$ according to Formulas 5, 6, and 7.
9: **else**
10: Update $\{\langle s_1, p_1\rangle, \langle s_2, p_2\rangle, \ldots, \langle s_m, p_m\rangle\}$ according to Formulas 8, 9, and 10.
11: **end_if**
12: **end_if**
13: **end_while**

D. MRs-centric Adaptive Metamorphic Testing

MRs-centric adaptive metamorphic testing (M-AMT) adds a feedback mechanism to control the execution process of traditional MT. First, M-AMT randomly selects an MR and a source test case belonging to a partition that is selected according to the test profile, generating the follow-up test case depend on the source test case, and then updates the test profile according to the result of test execution. Second, a partition is selected according to updated test profile, and an MR is randomly selected from the set of MRs whose source test case belong to selected partition.

Suppose that the input domain of SUT is divided into m ($m > 2$) partitions ($s_1, s_2, \ldots, s_m$). A set MR_{s_i} included metamorphic relations whose source test cases belong to the partition s_i, and $\mathcal{R} = \bigcup_{i=1}^{m} MR_{s_i}$ is a set that include all of metamorphic relations. The detailed M-AMT algorithm is given in Algorithm 3. In M-AMT, the first metamorphic relation MR is randomly selected from $\mathcal{R}$, and a source test case stc is randomly selected from a partition s_i related to the selected metamorphic relation, then the follow-up test case ftc is generate based on MR and stc, belonging to partition s_f ($f \in \{1, 2, \ldots, m\}$) (Lines 3 to 7 in Algorithm 3). If the results of stc and ftc violate the MR, there are following two situation, denoted δ_1, δ_2, respectively (Lines 8 to 12):

Situation 1 (δ_1): If $i = f$, then the test profile is updated according to Formulas 1 and 2.

Situation 2 (δ_2): If $i \neq f$, then the test profile is updated according to Formulas 5, 6, and 7.

Alternatively, if their results satisfy the MR, there are following two situation, denoted $\delta_1^{'}, \delta_2^{'}$, respectively (Lines 14 to 18):

Situation 1 ($\delta_1^{'}$): If $i = f$, then the test profile is updated according to Formulas 3 and 4.

Situation 2 ($\delta_2^{'}$): If $i \neq f$, then the test profile is updated according to Formulas 8, 9, and 10.

Next, M-AMT randomly selects a partition according to updated test profile, then a source test case is randomly selected from the selected partition and an metamorphic relation is randomly selected from a MR_{s_i} ($i \in \{1, 2, \ldots, m\}$) whose source test cases belong to the selected partition. On the basis of MR_{s_i}, the follow-up test case is generated from the source test case (Lines 22 and 23). After the execution of the source and follow-up test cases, their results are verified against the MR_{s_i}, then the test profile is updated (Lines 24 to 35). This process is repeated until a termination condition is satisfied (Line 2).

We developed a tool called AMTesting to the best of our knowledge. AMTesting has features such as termination condition setting, partition setting, test execution, and test report generation.

IV. Empirical Study

We have conducted a series of empirical studies to evaluate the performance of M-AMT. The design of experiments is described in this section.

A. Research Questions

RQ1 How efficient and effective is M-AMT at detecting faults? Fault-detection efficiency and effectiveness are key criterions for evaluating the performance of a testing technique. In our study, we chose three real-life programs, and applied mutation analysis to evaluate them.

RQ2 What is the actual test case selection overhead when using the M-AMT strategy? We empirically evaluate the test case generation overhead of M-AMT in detecting software faults.

B. Object Programs

We selected three real-life systems as the subject programs for our study: `China Unicom Billing System (CUBS)`, `Aviation Consignment Management System (ACMS)`, and `Expense Reimbursement System (ERS)`.

1) China Unicom Billing System (CUBS): CUBS provides an interface through which customers can know how much they need to pay according to plans, month charge, calls, and data usage. The details of two cell-phone plans are summarized in Tables I and II.

Algorithm 3 M-AMT

Input: $\epsilon, p_1, \ldots, p_m, MR_{s_1}, \ldots, MR_{s_m}, \mathcal{R}, counter$

Initialize $counter = 1$.
while termination condition is not satisfied **do**
  if $counter = 1$ **then**
    Randomly select an MR from $\mathcal{R}$, and $MR \in MR_{s_i}$.
    Increment counter by 1.
    Randomly select a source test case from the partition s_i, and generate the follow-up test case belonged to partition s_f $(f \in 1, 2, \ldots, m)$.
    Execute the source and follow-up test cases.
    if the results of source and follow-up test cases violate MR **then**
      if $i = f$ **then**
        Update the test profile according to Formulas 1 and 2.
      else
        Update the test profile according to Formulas 5, 6, and 7.
      end_if
    else
      if $i = f$ **then**
        Update the test profile according to Formulas 3 and 4.
      else
        Update the test profile according to Formulas 8, 9, and 10
      end_if
    end_if
  else
    Select a partition s_i according to the updated test profile, and then select an $MR^{'}$ from MR_{s_i}.
    Randomly select a source test case from the partition s_i, and generate the follow-up test case belonged to partition s_f $(f \in 1, 2, \ldots, m)$.
    Execute the source and follow-up test cases.
    if the results of source and follow-up test cases violate $MR^{'}$ **then**
      if $i = f$ **then**
        Update the test profile according to Formulas 1 and 2.
      else
        Update the test profile according to Formulas 5, 6, and 7.
      end_if
    else
      if $i = f$ **then**
        Update the test profile according to Formulas 3 and 4.
      else
        Update the test profile according to Formulas 8, 9, and 10
      end_if
    end_if
  end_if
end_while

TABLE I
PLAN A

Plan details		Month charge (CNY)			
		$option_1$	$option_2$	$option_3$	$option_4$
Basic	Free calls (min)	50	96	286	3000
	Free data (MB)	150	240	900	3000
Extra	Dialing calls (CNY/min)	0.25	0.15	0.15	0.15
	Data (CNY/KB)	0.0003			

TABLE II
PLAN B

Plan details		Month charge (CNY)			
		$option_1$	$option_2$	$option_3$	$option_4$
Basic	Free calls (min)	120	450	680	1180
	Free data (MB)	40	80	100	150
Extra	Dialing calls (CNY/min)	0.25	0.15	0.15	0.15
	Data (CNY/KB)	0.0003			

2) Aviation Consignment Management System (ACMS): ACMS aims to help airline companies check the allowance (weight) of free baggage, and the cost of additional baggage. Based on the destination, flights are categorised as either domestic or international. For international flights, the baggage allowance is greater if the passenger is a student (30kg), otherwise it is 20kg. Each aircraft offers three cabins classes from which to choose (economy, business, and first), with passengers in different classes having different allowances. The detailed price rules are summarized in Table III, where $price_0$ means economy class fare.

3) Expense Reimbursement System (ERS): ERS assists the sales Supervisor of a company with handling the following tasks: i) Calculating the cost of the employee who use the cars based on their titles and the number of miles actually traveled; ii) accepting the requests of reimbursement that include airfare, hotel accommodation, food and cell-phone expenses of the employee.

C. Variables

1) Independent Variables: The independent variable in our study is the testing technique. The M-AMT is the independent variable. In addition, we selected traditional MT that randomly selects source test cases as the baseline technique for comparison.

2) Dependent Variables: To answer our research question, F-measure [18] is used to evaluate the efficiency of MT and A-AMT strategies, which is defined as the expected number of test cases executions required to detect the first fault. In addition, to better illustrate the effectiveness improvement of M-AMT over traditional MT, N-measure, referred to the expected number of faults detected by a number of test cases that is preset before the test, is proposed. The N-measure is following the practical situation where there are not enough test resources to execute all test cases. In this study, we execute half, quarter, and quarter test cases of the CUBS, ACMS, and ERS, respectively (Note that the half test cases of ACMS and ERS have detected all faults).

TABLE III
ACMS BAGGAGE ALLOWANCE AND PRICING RULES

	Domestic flights			International flights		
	First class	Business class	Economy class	First class	Business class	Economy class
Carry on (kg)	5	5	5	7	7	7
Free checked-in (kg)	40	30	20	40	30	20/30
Additional baggage pricing (kg)	$price_0 * 1.5\%$			$price_0 * 1.5\%$		

TABLE IV
THE INFORMATION OF STUDIED PROGRAMS

program	Total Number of Mutants	Number of Selected Mutants	Number of MRs	Number of Partitions
CUBS	210	9	142	4
ACMS	187	4	735	8
ERS	180	4	1130	12

For RQ2, an obvious metric is the time required to select test cases. In this study, corresponding to the F-measure and N-measure, we used the F-time and N-time to measure the time to select test cases for detecting the first fault and select a preset number of test cases, respectively.

For F-measure, F-time, and N-time, a smaller value intuitively implies a better performance. Inversely, a bigger value of N-measure indicates a better performance.

D. Experimental Settings

1) Faults: We introduced artificial faults into the studied programs using mutation analysis [28]–[31]. More specifically, we used the tool muJava [32] to create 577 faulty versions (i.e., mutants) of our programs, where each mutant was created from the original programs by applying a syntactic change to its source code. Each syntactic change is determined by a soCcalled mutation operator. After removing equivalent mutants, we then also deleted mutants that were too easily detected — removing mutants that could be detected with less than 20 randomly generated test cases. Table IV (the second and third columns) shows the basic information of the used programs and their mutants.

2) Metamorphic Relations: In our study, to obtain metamorphic relations, we made use of METRIC [33], which is based on the category–choice framework [34]. In METRIC, only two distinct complete test frames that are abstract test cases defining possible combinations of inputs, are considered by the tester to generate an MR. In general, METRIC has the following steps to identify MRs:

Step1. Users select two relevant and distinct complete test frames as a *candidate pair*.

Step2. Users determine whether or not the selected candidate pair is useful for identifying an MR, and if it is, then provide the corresponding MR description.

Step3. Restart from step 1, and repeat until all candidate pairs are exhausted, or the predefined number of MRs to be generated is reached.

Following the above guidelines, we identified MRs for studied programs and the number metamorphic relations of programs is summarized in Table IV (the fourth column).

3) Partitioning: We used the category–partition method (CPM) [34], a typical PT technique based on *Specified As Equivalent*, to conduct the partitioning. In CPM, a functional requirement is decomposed into a set of categories, which are major properties or characteristics of the input parameters or environment conditions. Each category is further divided into disjoint choices, which include all the different kinds of values that are possible for the category. As a consequence, we obtained categories and choices for every studied programs. With those categories\choices and constraints among choices, we further generated a set of complete test frames and each of them corresponds to a partition. Note that the number of partitions may vary with the granularity level. In our study, we only consider two categories to conduct partition, and the results of partitioning is recorded in Table IV (the fifth column).

4) Initial Test Profile: Because we do not know the distribution of faults before testing, a feasible method is to use a uniform probability distribution as the initial test profile. On the other hand, testers may also use past experience to guide a different probability distribution as the initial profile.

5) Constants: Previous studies [35], [36] set a relatively small DRT parameter ϵ (e.g. $\epsilon = 0.05$). We followed these studies to set $\epsilon = 0.05$.

E. Experimental Environment

Our experiments were conducted on a virtual machine running the Ubuntu 18.04 64-bit operating system, with two CPUs, and a memory of 4GB. The test scripts were written in Java. To ensure statistically reliable values of the metrics (F-measure, N-measure, F-time, N-time), each testing session was repeated 30 times with 30 different seeds following the guidelines proposed in [37], and the average value calculated.

V. EXPERIMENTAL RESULTS

1) RQ1: Fault Detection Effectiveness: Tables V shows the experimental results of F-measure and N-measure, and the distribution of them on each object program are displayed by the boxplots in Figures 2 and 3. In the boxplot, the upper and lower bounds of the box represent the third and first quartiles of a metric, respectively, and the middle line denotes the median value. The upper and lower whiskers respectively indicate the largest and smallest data within the range of $\pm 1.5 \times IQR$, where IQR is the interquartile range. The solid circle represents the mean value of a metric.

From Table V and Figures 2, and 3, we have the following observations:

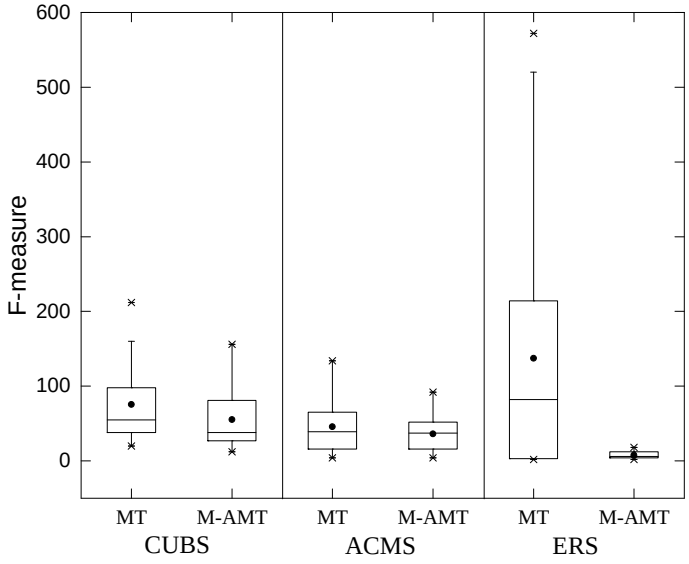

Fig. 2. F-measure Boxplots for Each Program

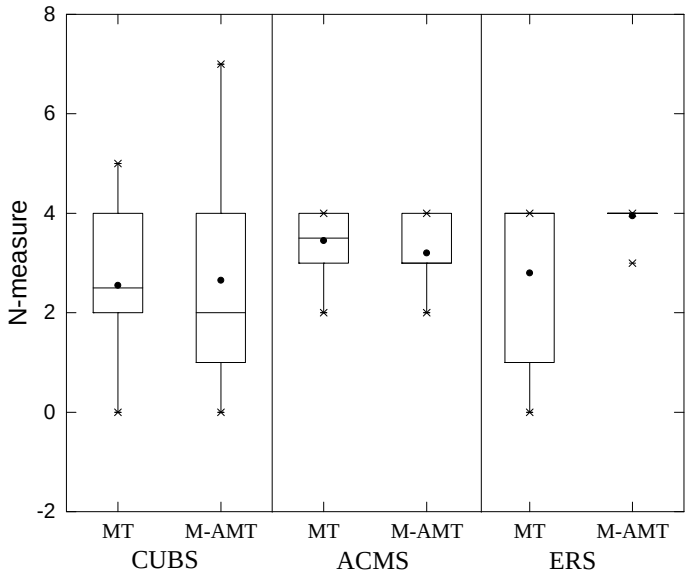

Fig. 3. N-measure Boxplots for Each Program

TABLE V
F-MEASURE AND N-MEASURE FOR STUDIED PROGRAMS

program	Strategy	F-measure	N-measure
CUBS	MT	75.60	2.55
	M-AMT	55.30	2.65
ACMS	MT	45.80	3.45
	M-AMT	36.10	3.20
ERS	MT	139.50	2.80
	M-AMT	7.70	3.95

TABLE VI
F-TIME AND N-TIME FOR STUDIES PROGRAMS (IN MS)

program	Strategy	F-time	N-time
CUBS	MT	5.76	10.24
	M-AMT	28.45	71.70
ACMS	MT	3.79	15.24
	M-AMT	28.85	137.00
ERS	MT	29.05	4.15
	M-AMT	4.10	51.60

1 MT executes test cases in a random manner, that is, MT randomly selects test cases from input domain as source test cases, which does not use of any information returned from the testing process. Based on MT, M-AMT introduces software cybernetics [19] into MT, analyzing the current test results to select "better" test cases that is more likely to detect faults.
2 In terms of F-measure, M-AMT need fewer test cases to detect the first fault for all three programs compared to the MT.
3 Executing the same number of test cases, M-AMT reveals more faults than MT for all three programs.

2) RQ2: Selection Overhead: The detailed results of F-time and N-time are summarized in the Table VI, on the basis of which, we have the following observations:

1 *the performance of the test techniques:* The traditional MT that randomly selects test cases and executes them. M-AMT first adjusts test profile according to current test results, then selects test cases and executes them. Our results indicate that MT selection test cases consumes less time than M-AMT.
2 *In terms of F-time and N-time:* In most of scenarios (except for F-time of ERS), MT selection test cases consumes less time than M-AMT. M-AMT require additional computation compared to MT and such additional computation is compensated by having fewer program executions. When the test case execution time saved by M-AMT is sufficient to cover the additional computation, M-AMT will have a better performance. To detect the first fault of ERS, M-AMT requires far fewer test cases than MT, hence M-AMT selection test cases consumes less time than MT.

In summary, M-AMT was shown to be the better testing technique than MT across F- and N-measure, which means that The fault detection efficiency of M-AMT is higher than MT. Meanwhile, M-AMT takes extra time to adjust the test profile and select a partition, while such additional time can be compensated by having fewer programs executions. This further indicates that between the proposed M-AMT and MT, M-AMT should be used.

VI. RELATED WORK

In this section, we describe related work from two perspectives: i) related to MT; ii) related to improving RT and PT.

A. Metamorphic Testing

When testing a software system, the oracle problem appears in some situations where either an oracle does not exist for

the tester to verify the correctness of the computed results; or an oracle does exist but cannot be used. The oracle problem often occurs in software testing, which renders many testing techniques inapplicable [2]. To alleviate the oracle problem, Chen et al. [5] proposed a technique named metamorphic testing (MT) that has been receiving increasing attention in the software testing community [2], [6], [17]. The main contributions to MT in the literature focused on the following aspects: i) MT theory; ii) combination with other techniques; iii) application of MT.

1 *Theoretical development of MT:* The MRs and the source test cases are the most important components of MT. However, defining MRs can be difficult. Chen et al. [33] proposed a specification-based method and developed a tool called MR-GENerator for identifying MRs based on category-choice framework [34]. Zhang et al. [38] proposed a search-based approach to automatic inference of polynomial MRs for a software under test, where a set of parameters is used to represent polynomial MRs, and the problem of inferring MRs is turn into a problem of searching for suitable values of the parameters. Then, particle swarm optimization is used to solve the search problem. Sun et al. [39] proposes a data-mutation directed metamorphic relation acquisition methodology, in which data mutation is employed to construct input relations and the generic mapping rule associated with each mutation operator to construct output relations. Liu et al. [40] proposed to systematically construct MRs based on some already identified MRs.
Without doubt, "good" MRs can improve the fault detection efficiency of MT. Chen et al. [9] reported that good MRs are those that can make the execution of the source-test case as different as possible to its follow-up test case. This perspective has been confirmed by the later studies [15], [16]. Asrafi et al. [41] conduct a case study to analyze the relationship between the execution behavior and the fault-detection effectiveness of metamorphic relations by code coverage criteria, and the results showed a strong correlation between the code coverage achieved by a metamorphic relation and its fault-detection effectiveness.
Source test cases also have a important impact on the fault detection effectiveness of MT. Chen et al. [11] compared the effects of source test cases generated by special value testing and random testing on the effectiveness of MT, and found that MT can be used as a complementary test method to special value testing. Batra and Sengupta [15] integrated genetic algorithms into MT to select source test cases maximising the the paths traversed in the software under test. Dong et al. [16] proposed a Path–Combination–Based MT method that first generates symbolic input for each executable paths and minis relationships among these symbolic inputs and their outputs, then constructs MRs on the basis of these relationships, and generates actual test cases corresponding to the symbolic inputs.
Different from the above investigates, we focused on performing test cases and MRs with fault revealing capabilities as quickly as possible by making use of feedback information. We first divided the input domain into disjoint partitions, and randomly selected an MR to generate follow-up test cases depended on source test case of related input partitions, then updated the test profile of input partitions according to the results of test execution. Next, a partition was selected according to updated test profile, and an MR was randomly selected from the set of MRs whose source test cases belong to selected partition.

2 *Combination with other techniques:* In order to improve the applicability and effectiveness of MT, it has been integrated into other techniques. Xie et al. [42] combined the MT with the spectrum-based fault localization (SBFL), extend the application of SBFL to the common situations where test oracles do not exist. Dong et al. [43] proposed a method for improving the efficiency of evolutionary testing (ET) by considering MR when fitness function is constructed. Liu et al. [44] introduced MT into fault tolerance and proposed a theoretical framework of a new technique called Metamorphic Fault Tolerance (MFT), which can handle system failure without the need of oracles during failure detection. In MFT, the trustworthiness of a test case depends on the number of violations or satisfactions of metamorphic relations. The more relations are satisfied and the less relations are violated, the more trustable test case is.

3 *Application of MT:* Sun et al. [12], [45] proposed a metamorphic testing framework for web services taking into account the unique features of SOA, in which MRs are derived from the description or Web Service Description Language (WSDL) [12] of the Web service, and on the basis on of MRs, follow-up test cses are generated depended on source test cases that are randomly generated according to the WSDL. Segura et al. [46] present a metamorphic testing approach for the detection of faults in RESTful Web APIs where they proposed six abstract relations called Metamorphic Relation Output Patterns (MROPs) that can then be instantiated into one or more concrete metamorphic relations. To evaluate this approach, they used both automatically seeded and real faults in six subject Web APIs.

B. Dynamic Random Testing

Cai et al. introduced software cybernetics [19] to software testing, and proposed Adaptive Testing (AT) [47]–[49], which takes advantage of feedback information to control the execution process, has been shown that AT outperforms RT in terms of T-measure and increasing the number of faults, which means that AT has higher efficiency and effectiveness than RT. However, AT may require a very long execution time in practice. To alleviate this, Cai et al. [20] proposed DRT, which uses testing information to dynamically adjust the test profile. There are several things that can impact on DRTs test efficiency. Yang et al. [50] proposed A-DRT, which adjusts parameters during the testing process. Li et al. [36] developed O-DRT, which has an objective function and a pre-defined parameter f. During the testing process, if the value of the objective function is greater than f, the test profile is adjusted

to a theoretically optimal one. Lv et al. [51] introduced two parameters for adjusting probability of different partitions, namely ϵ for adjusting probability of partitions where a test case detects a fault and δ for adjusting probability of partitions where a test case does not detect a fault, and ϵ should be larger than δ. Furthermore, they provided the guidance of setting ϵ and δ by an interval of ϵ/δ. Based on DRT, Sun et al. [18] proposed a new technique named adaptive partition testing to increase the adaptability of DRT itself, where the test profile and the probability adjusting factors are dynamically updated. Furthermore, they develop two algorithms, Markov-chain based adaptive partition testing and reward-punishment based adaptive partition testing, to implement the proposed approach.

VII. Conclusion

Metamorphic testing (MT) is a promising technique that alleviates the oracle problem. To improve the efficiency of MT, the most of researchers focused on generating and selecting the "better" metamorphic relations (the execution results of the test cases are more likely to violate them), ignoring the impact of the source test cases. Differently, based on the partition testing, we proposed an adaptive metamorphic testing technique that aim to control the execution process of MT and make use of the feedback information, improving the fault detection efficiency of MT. Furthermore, We implement the proposed technique in APT2MT, to the best of our knowledge. Empirical studies have been conducted to evaluate the performance of M-AMT and MT using three real-life programs associated with 17 distinct faults that covered different types of mutation operators and whose detection needs more than 20 randomly generated test cases. It has been shown that M-AMT could use significantly fewer test cases to detect the first fault than MT. Moreover, Executing the same number of test cases, M-AMT detected more faults than MT.

In our future work, we plan to conduct experiments on more real-life programs to further validate the effectiveness of M-AMT, and identify the limitations of our approach.

Acknowledgment

This research is supported by the Beijing Natural Science Foundation (Grant No. 4162040), the National Natural Science Foundation of China (Grant No. 61370061), the Aeronautical Science Foundation of China (Grant No. 2016ZD74004), and the Fundamental Research Funds for the Central Universities (Grant No. FRF-GF-17-B29).

References

[1] E. J. Weyuker, "On testing non-testable programs," *The Computer Journal*, vol. 25, no. 4, pp. 465–470, 1982.

[2] E. T. Barr, M. Harman, P. McMinn, M. Shahbaz, and S. Yoo, "The oracle problem in software testing: A survey," *IEEE Transactions on Software Engineering*, vol. 41, no. 5, pp. 507–525, 2015.

[3] K. Patel and R. M. Hierons, "A mapping study on testing non-testable systems," *Software Quality Journal*, vol. 26, no. 4, pp. 1373–1413, 2018.

[4] S. S. Brilliant, J. C. Knight, and P. Ammann, "On the performance of software testing using multiple versions," in *Proceedings of the 20th International Symposium on Fault-Tolerant Computing (FTCS'90)*, 1990, pp. 408–415.

[5] T. Y. Chen, S. C. Cheung, and S. M. Yiu, "Metamorphic testing: A new approach for generating next test cases," Department of Computer Science , Hong Kong University of Science and Technology, Hong Kong, Tech. Rep. HKUST-CS98-01, Tech. Rep., 1998.

[6] T. Y. Chen, F.-C. Kuo, H. Liu, P.-L. Poon, D. Towey, T. Tse, and Z. Q. Zhou, "Metamorphic testing: A review of challenges and opportunities," *ACM Computing Surveys*, vol. 51, no. 1, p. 4, 2018.

[7] K. Y. Sim, C. S. Low, and F.-C. Kuo, "Eliminating human visual judgment from testing of financial charting software," *Journal of Software*, vol. 9, no. 2, pp. 298–312, 2014.

[8] W. K. Chan, S.-C. Cheung, J. C. Ho, and T. Tse, "Pat: A pattern classification approach to automatic reference oracles for the testing of mesh simplification programs," *Journal of Systems and Software*, vol. 82, no. 3, pp. 422–434, 2009.

[9] T. Y. Chen, D. Huang, T. Tse, and Z. Q. Zhou, "Case studies on the selection of useful relations in metamorphic testing," in *Proceedings of the 4th lberoAmerican Symposium on Software Engineering and Knowledge Engineer-ing (JIISIC'04)*, 2004, pp. 569–583.

[10] Y. Cao, Z. Q. Zhou, and T. Y. Chen, "On the correlation between the effectiveness of metamorphic relations and dissimilarities of test case executions," in *Proceedings of the 13th International Conference on Quality Software (QSIC'13)*, 2013, pp. 153–162.

[11] T. Y. Chen, F.-C. Kuo, Y. Liu, and A. Tang, "Metamorphic testing and testing with special values." in *Proceedings of the 5th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD'04)*, 2004, pp. 128–134.

[12] C.-A. Sun, G. Wang, B. Mu, H. Liu, Z. Wang, and T. Y. Chen, "Metamorphic testing for web services: Framework and a case study," in *Proceedings of the 9th IEEE International Conference on Web Services (ICWS'11)*, 2011, pp. 283–290.

[13] T. Chen, P. Poon, and X. Xie, "Metric: Metamorphic relation identification based on the category-choice framework," *Journal of Systems and Software*, vol. 116, pp. 190–14, 2014.

[14] X. Xie, J. Li, C. Wang, and T. Y. Chen, "Looking for an mr? try metwiki today," in *Proceedings of the 1st International Workshop on Metamorphic Testing (MET'16), Co-located with the 38th International Conference on Software Engineering (ICSE'16)*, 2016, pp. 1–4.

[15] G. Batra and J. Sengupta, "An efficient metamorphic testing technique using genetic algorithm," in *International Conference on Information Intelligence, Systems, Technology and Management*, 2011, pp. 180–188.

[16] G. Dong, T. Guo, and P. Zhang, "Security assurance with program path analysis and metamorphic testing," in *Proceedings of the 4th IEEE International Conference on Software Engineering and Service Science (ICSESS'13)*, 2013, pp. 193–197.

[17] S. Segura, G. Fraser, A. B. Sanchez, and A. Ruiz-Cortés, "A survey on metamorphic testing," *IEEE Transactions on Software Engineering*, vol. 42, no. 9, pp. 805–824, 2016.

[18] C.-A. Sun, H. Dai, H. Liu, T. Y. Chen, and K.-Y. Cai, "Adaptive partition testing," *IEEE Transactions on Computers*, 2018.

[19] K.-Y. Cai, "Optimal software testing and adaptive software testing in the context of software cybernetics," *Information and Software Technology*, vol. 44, no. 14, pp. 841–855, 2002.

[20] K. Cai, H. Hu, C.-h. Jiang, and F. Ye, "Random testing with dynamically updated test profile," in *Proceedings of the 20th International Symposium On Software Reliability Engineering (ISSRE'09)*, 2009, pp. 1–2.

[21] H. Liu, F.-C. Kuo, D. Towey, and T. Y. Chen, "How effectively does metamorphic testing alleviate the oracle problem?" *IEEE Transactions on Software Engineering*, vol. 40, no. 1, pp. 4–22, 2014.

[22] E. J. Weyuker and B. Jeng, "Analyzing partition testing strategies," *IEEE Transactions on Software Engineering*, vol. 17, no. 7, pp. 703–711, 1991.

[23] K.-Y. Cai, T. Jing, and C.-G. Bai, "Partition testing with dynamic partitioning," in *Proceedings of the 29th International Computer Software and Applications Conference (COMPSAC'05)*, 2005, pp. 113–116.

[24] T. Y. Chen and Y.-T. Yu, "On the relationship between partition and random testing," *IEEE Transactions on Software Engineering*, vol. 20, no. 12, pp. 977–980, 1994.

[25] ——, "On the expected number of failures detected by subdomain testing and random testing," *IEEE Transactions on Software Engineering*, vol. 22, no. 2, pp. 109–119, 1996.

[26] P. E. Ammann and J. C. Knight, "Data diversity: An approach to

software fault tolerance," *IEEE Transactions on Computers*, no. 4, pp. 418–425, 1988.

[27] G. B. Finelli, "Nasa software failure characterization experiments," *Reliability Engineering & System Safety*, vol. 32, no. 1-2, pp. 155–169, 1991.

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

[29] J. Chen, L. Zhu, T. Y. Chen, D. Towey, F.-C. Kuo, R. Huang, and Y. Guo, "Test case prioritization for object-oriented software: An adaptive random sequence approach based on clustering," *Journal of Systems and Software*, vol. 135, pp. 107–125, 2018.

[30] C. Mao, T. Y. Chen, and F.-C. Kuo, "Out of sight, out of mind: A distance-aware forgetting strategy for adaptive random testing," *Science China Information Sciences*, vol. 60, no. 9, pp. 092 106:1–092 106:21, 2017.

[31] J. Chen, F.-C. Kuo, T. Y. Chen, D. Towey, C. Su, and R. Huang, "A similarity metric for the inputs of oo programs and its application in adaptive random testing," *IEEE Transactions on Reliability*, vol. 66, no. 2, pp. 373–402, 2017.

[32] Y.-S. Ma, J. Offutt, and Y. R. Kwon, "Mujava: an automated class mutation system," *Software Testing, Verification and Reliability*, vol. 15, no. 2, pp. 97–133, 2005.

[33] T. Y. Chen, P.-L. Poon, and X. Xie, "Metric: Metamorphic relation identification based on the category-choice framework," *Journal of Systems and Software*, vol. 116, pp. 177–190, 2016.

[34] T. J. Ostrand and M. J. Balcer, "The category-partition method for specifying and generating fuctional tests," *Communications of the ACM*, vol. 31, no. 6, pp. 676–686, 1988.

[35] Z. Yang, B. Yin, J. Lv, K. Cai, S. S. Yau, and J. Yu, "Dynamic random testing with parameter adjustment," in *Proceedings of the 6th IEEE International Workshop on Software Test Automationthe, Co-located with the 38th IEEE Annual International Computer Software and Applications Conference (COMPSAC14)*, 2014, pp. 37–42.

[36] Y. Li, B.-B. Yin, J. Lv, and K.-Y. Cai, "Approach for test profile optimization in dynamic random testing," in *Proceedings of the 39th International Computer Software and Applications Conference (COMPSAC15)*, vol. 3, 2015, pp. 466–471.

[37] A. Arcuri and L. Briand, "A practical guide for using statistical tests to assess randomized algorithms in software engineering," in *Proceedings of the 33rd International Conference on Software Engineering (ICSE'11)*, 2011, pp. 1–10.

[38] J. Zhang, J. Chen, D. Hao, Y. Xiong, B. Xie, L. Zhang, and H. Mei, "Search-based inference of polynomial metamorphic relations," in *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering (ASE'14)*, 2014, pp. 701–712.

[39] C.-A. Sun, Y. Liu, Z. Wang, and W. K. Chan, "μmt: A data mutation directed metamorphic relation acquisition methodology," in *Proceedings of the 1st International Workshop on Metamorphic Testing (MET'16), Co-located with the 38th International Conference on Software Engineering (ICSE'16)*, 2016, pp. 12–18.

[40] H. Liu, X. Liu, and T. Y. Chen, "A new method for constructing metamorphic relations," in *Proceedings of the 12th International Conference on Quality Software (QSIC'12)*, 2012, pp. 59–68.

[41] M. Asrafi, H. Liu, and F.-C. Kuo, "On testing effectiveness of metamorphic relations: A case study," in *Proceedings of the 15th International Conference on Secure Software Integration and Reliability Improvement (SSIRI'11)*, 2011, pp. 147–156.

[42] X. Xie, W. E. Wong, T. Y. Chen, and B. Xu, "Metamorphic slice: An application in spectrum-based fault localization," *Information and Software Technology*, vol. 55, no. 5, pp. 866–879, 2013.

[43] G. Dong, S. Wu, G. Wang, T. Guo, and Y. Huang, "Security assurance with metamorphic testing and genetic algorithm," in *Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT'10)*, vol. 3, 2010, pp. 397–401.

[44] H. Liu, I. I. Yusuf, H. W. Schmidt, and T. Y. Chen, "Metamorphic fault tolerance: An automated and systematic methodology for fault tolerance in the absence of test oracle," in *Proceedings of the 36th International Conference on Software Engineering (ICSE'14)*, 2014, pp. 420–423.

[45] C.-A. Sun, G. Wang, B. Mu, H. Liu, Z. Wang, and T. Y. Chen, "A metamorphic relation-based approach to testing web services without oracles," *International Journal of Web Services Research*, vol. 9, no. 1, pp. 51–73, 2012.

[46] S. Segura, J. A. Parejo, J. Troya, and A. Ruiz-Cortés, "Metamorphic testing of restful web apis," *IEEE Transactions on Software Engineering*, vol. 44, no. 11, pp. 1083–1099, 2018.

[47] K.-Y. Cai, B. Gu, H. Hu, and Y.-C. Li, "Adaptive software testing with fixed-memory feedback," *Journal of Systems and Software*, vol. 80, no. 8, pp. 1328–1348, 2007.

[48] H. Hu, W. E. Wong, C.-H. Jiang, and K.-Y. Cai, "A case study of the recursive least squares estimation approach to adaptive testing for software components," in *Proceedings of the 5th International Conference on Quality Software (QSIC'05)*, 2005, pp. 135–141.

[49] H. Hu, C.-H. Jiang, and K.-Y. Cai, "An improved approach to adaptive testing," *International Journal of Software Engineering and Knowledge Engineering*, vol. 19, no. 05, pp. 679–705, 2009.

[50] Z. Yang, B. Yin, J. Lv, K.-Y. Cai, S. S. Yau, and J. Yu, "Dynamic random testing with parameter adjustment," in *Proceedings of the 38th IEEE Annual International Computer Software and Applications Conference Workshops (COMPSACW'14)*, 2014, pp. 37–42.

[51] J. Lv, H. Hu, and K.-Y. Cai, "A sufficient condition for parameters estimation in dynamic random testing," in *Proceedings of the 35th IEEE Annual International Computer Software and Applications Conference Workshops (COMPSACW'11)*, 2011, pp. 19–24.