Identifying metamorphic relations
A data mutation directed approach

Sun, Chang-ai; Jin, Hui; Wu, SiYi; Fu, An; Wang, ZuoYi; Chan, Wing Kwong

Published in:
Software - Practice and Experience

Published: 01/03/2024

Document Version:
Post-print, also known as Accepted Author Manuscript, Peer-reviewed or Author Final version

Published version (DOI):
10.1002/spe.3280

Publication details:
Sun, C., Jin, H., Wu, S., Fu, A., Wang, Z., & Chan, W. K. (2024). Identifying metamorphic relations: A data mutation directed approach. *Software - Practice and Experience*, *54*(3), 394-418. https://doi.org/10.1002/spe.3280

Received: Added at production | Revised: Added at production | Accepted: Added at production

DOI: xxx/xxxx

EXTENDED CONFERENCE PAPER

Identifying Metamorphic Relations: A Data Mutation Directed Approach

Chang-ai Sun*[1] | Hui Jin[1] | SiYi Wu[1] | An Fu[1] | ZuoYi Wang[1] | Wing Kwong Chan[2]

[1]School of Computer and Communication Engineering, University of Science and Technology Beijing, Beijing, China
[2]Department of Computer Science, City University of Hong Kong, Hong Kong, China

Correspondence
*Corresponding author. Email: casun@ustb.edu.cn

Present Address
A preliminary version of this paper[1] was published in the *Proceedings of the 1st International Workshop on Metamorphic Testing* (MET 2016), in conjunction with the 38th International Conference on Software Engineering (ICSE 2016).

Summary

Metamorphic testing (MT) is an effective technique to alleviate the test oracle problem. The principle of MT is to detect failures by checking whether some necessary properties, commonly known as metamorphic relations (MRs), of software under test (SUT) hold among multiple executions of source and follow-up test cases. Since both the generation of follow-up test cases and test result verification depend on MRs, the identification of MRs plays a key role in MT, which is an important yet difficult task requiring deep domain knowledge of the SUT. Accordingly, techniques that can direct a tester to identify MRs effectively are desirable. In this paper, we propose μMT, a data mutation directed approach to identifying MRs. μMT guides a tester to identify MRs by providing a set of data mutation operators and template-style mapping rules, which not only alleviates the difficulties faced in the process of MR identification but also improves the identification effectiveness. We have further developed a tool to implement the proposed approach and conducted an empirical study to evaluate the MR identification effectiveness of μMT and the performance of MRs identified by μMT with respect to fault detection capability and statement coverage. The empirical results show that μMT is able to identify MRs for numeric programs effectively, and the identified MRs have high fault detection capability and statement coverage. The work presented in this paper advances the field of MT by providing a simple yet practical approach to the MR identification problem.

KEYWORDS:
Software Testing, Test Oracle, Metamorphic Testing, Metamorphic Relation

1 | INTRODUCTION

Software testing is a widely used software quality assurance approach [2]. A general software testing process executes the software under test (SUT) using test cases and checks the corresponding outputs against the test oracle. Unfortunately, in many cases, test oracle does not exist or is too costly to apply, which is commonly known as the test oracle problem[3,4,5]. For instance, consider the program $\sin(X)$ that accepts an angle value in degree X and computes its sine value. Deciding the expected values for some specific inputs, such as 30° and 90°, may be easy. However, deciding the expected values for arbitrary angle values, such as 37°, becomes difficult without a test oracle, which makes a tester unable to judge whether the test passes or not. As a result, many testing techniques become hard to be practiced when the test oracle problem exists.

To alleviate the test oracle problem, researchers have proposed a variety of approaches, such as N-version programming[6], assertion[7], and metamorphic testing (MT)[8,9]. Among them, MT is well known for its simple concept, straightforward implementation, and low cost in result verification. The underlying idea of MT is to detect faults by checking whether a property of the SUT is violated. The property is hereby manifested as a kind of metamorphic relations (MRs) held among multiple executions. As an alternative view, an MR may be represented as a relation among software inputs and their corresponding outputs. Accordingly, a failure is said to be detected when an MR is violated. In this way, MT can detect failures without the need for test oracles, thus alleviating the test oracle problem.

Since its inception, MT has increasingly received attention, and significant advances have been made in recent years. Two fundamental issues of MT are the generation and selection of effective source test cases and the identification and selection of MRs, respectively[10]. For the former, various test case generation or selection techniques were proposed, such as genetic algorithm-based test case generation[11], path-combination-based test case generation[12], adaptive random testing (ART) based test case selection[13,14], dynamic symbolic execution based test case generation[15,16], and iterative MT[17,10]. For the latter, representative approaches include search-based MR inference[18], graph-kernel based MR prediction[19], machine learning-based MR identification[20], pattern-based MR identification[21], data mutation-based MR identification[22], category choice framework based MR identification[23,24], and MR composition[25,26,27]. Furthermore, MT has been adopted to test applications in a variety of domains and found real-life faults, including cybersecurity[28], autonomous car driving systems[29,30,31], embedded systems[32,33,34], Web services[35,36,37], image processing applications[38], machine learning applications[39,40,41], artificial intelligence applications[42], compilers[43,44,45,46], search engines[47,48,49], RESTful Web APIs[50], and health care simulation systems[51].

While significant progress has been made, some issues deserve substantial efforts, particularly those related to MR identification[52,53]. Despite that several MRs identification techniques have been proposed, most MRs are still manually identified based on the tester's domain knowledge and previous experience[54]. Accordingly, concrete guidelines are desirable to direct testers to effectively and efficiently identify MRs. Inspired by data mutation (DM) that generates new test cases by mutating existing ones according to some pre-defined rules, namely data mutation operators (DMOs)[55], we leverage such operators to guide a tester to consider the relationship between two inputs (i.e., the input relation of MR), with which the tester can further predict possible variations of their outputs (i.e., the output relation of MR) according to the necessary properties of the software under test. In this way, the commonalities between DM and MT motivate us to propose an efficient way to derive MRs based on DM. We have proposed a data mutation-directed MR identification approach in the preliminary version[1] of this paper, which summarized five data mutation operators (DMOs). In this work, we take a step forward to improve our proposed approach further and comprehensively evaluate its effectiveness and efficiency. Specifically, we significantly extend the conference work from the following aspects: (i) more DMOs are proposed, including two additional DMOs and eight composite DMOs; (ii) refined mapping rules are proposed for efficiently deriving the output relation of an MR; (iii) three additional subject programs are involved in the empirical study, while more metrics and additional baseline techniques are used for evaluation.

The main contributions of this paper, together with its preliminary version[1], are as follows:

1. A data mutation-directed MR identification approach called μMT is proposed. The proposed approach involves seven data mutation operators (DMOs), eight composite DMOs, and two sets of mapping DMOs rules for output relations of MRs, providing a clear clue to direct the tester to identify MRs effectively.
2. A prototype tool has been developed to support μMT, which further improves MR identification's efficiency.
3. An empirical study has been conducted to comprehensively evaluate μMT in terms of the MR identification effectiveness and the performance of identified MRs using μMT with respect to fault detection and statement coverage.

The rest of this paper is structured as follows. Section 2 introduces the underlying concepts of MT, MR composition, and data mutation. Section 3 presents the proposed methodology, including its motivation, key techniques, and illustration. The prototype tool is presented in Section 4. Section 5 discusses the settings of the empirical study, while the results are reported in Section 6. Section 7 reviews related work. Section 8 concludes the paper.

2 | BACKGROUND

This section describes the underlying concepts of metamorphic testing, metamorphic relation composition, and data mutation.

2.1 | Metamorphic Testing (MT)

MT was first proposed as a technique for generating new test cases using existing ones that did not reveal software failure[8]. It was soon realized that MT is also an effective testing technique for addressing the test oracle problem. A key element in MT is metamorphic relation (MR), which, in a simple form, consists of two parts: a relation among multiple inputs and a relation among multiple corresponding outputs. A specialized kind of MR is defined as follows.

Given program P that implements the function f. Let R and R_f are a sequence of inputs $(x_1, x_2, \ldots, x_n)$ where $(n >= 1)$ and their expected outputs $(f(x_1), f(x_2), \ldots, f(x_n))$, respectively. An MR of P is a necessary property of f over R and R_f, denoted as (R, R_f), such that $(x_1, x_2, \ldots, x_n) \in R$ implies $(f(x_1), f(x_2), \ldots, f(x_n)) \in R_f$, as shown in Eq. 1.

$$R(x_1, \ldots, x_n) \Rightarrow R_f(f(x_1), \ldots, f(x_n)) \tag{1}$$

Note that Eq. 1 can be extended into a general form of MR that follow-up test cases can be derived with the information of the outputs of source inputs, and R_f involves both source and follow-up test cases. In this paper, we restrict the scope of our work to the follow-up test case construction process independent of the execution results of source test cases. For instance, x_1 and x_2 are two inputs of P and satisfy the input relation $R(x_1, x_2)$, and their corresponding outputs, denoted as $P(x_1)$ and $P(x_2)$, respectively, do not satisfy $R_f(P(x_1), P(x_2))$. In this situation, the MR is violated, which indicates the detection of a fault. MRs play a key role in verifying test results, and the expected outputs of individual inputs are no longer necessary in the context of MT. Furthermore, the input relation in an MR also provides a way to generate more follow-up test cases.

Figure 1 illustrates the baseline methodology of MT, consisting of the following steps:

1) *Identification of MRs*: MRs are identified from the specification of SUT, which requires deep knowledge of the domain related to SUT.

2) *Generation of source test cases*: Existing techniques, such as special values and random testing, can be used for this purpose.

3) *Generation of follow-up test cases according to MRs*: Follow-up test cases are generated based on source test cases through the applicable MRs. For a given MR, a source test case and its corresponding follow-up test case form a metamorphic test group (MTG).

4) *Test Execution*: Both source test cases and follow-up test cases are used to execute the SUT, and their corresponding outputs are collected accordingly.

5) *Verification of test results*: It checks whether the applied MR holds among the actual outputs of source and follow-up test case. If this is the case, the test passes; otherwise, a fault is said to be detected.

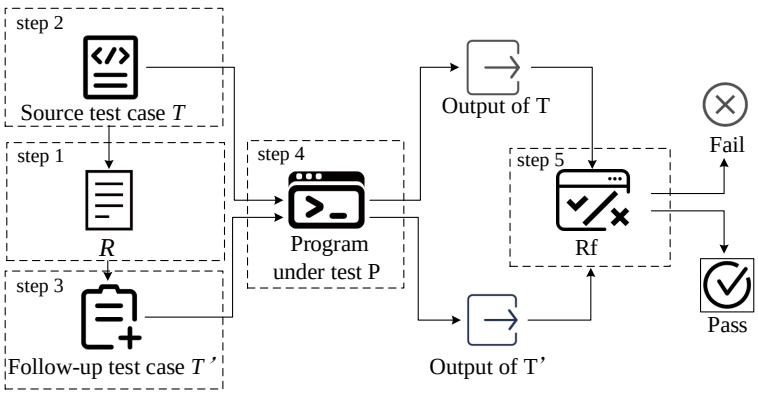


Figure 1 Principle of Metamorphic Testing

We here use an example to illustrate the above steps of the MT methodology. Suppose P is a program to compute the shortest path of two nodes in an undirected graph G. If the structure of G is too complex, it becomes hard to mathematically derive

the shortest path from a to b. Let $P(G, a, b)$ denote the shortest path from node a to b in G. Based on the properties of the undirected graph, we can acquire two MRs as follows: (i) when swapping the starting node and ending node, the shortest distance between two nodes should be the same (MR_1); (ii) when the graph is permutated, the result of the calculations should remain unchanged (MR_2). According to MR_1, given a source test case (G, a, b), a follow-up test case (G, b, a) can be generated. Testing is performed by executing P with these two test cases and checking whether $|P(G, a, b)| = |P(G, b, a)|$ is satisfied. If it does not hold, a fault is detected. For MR_2, a follow-up test case $(G^{'}, a^{'}, b^{'})$ is derived according to the source test case (G, a, b), where $G^{'}$ is a permutation of G, $a^{'}$ and $b^{'}$ is the permutation counterparts of a and b, respectively. After executing P with (G, a, b) and $(G^{'}, a^{'}, b^{'})$, we check whether $|P(G^{'}, a^{'}, b^{'})| = |P(G, a, b)|$. Similarly, the violation of output relation in MR_2 indicates the detection of a fault.

MRs are the key component of MT since they generate follow-up test cases and provide a mechanism for verifying test results.

2.2 | Metamorphic Relation Composition

The MR composition technique aims at constructing new MRs from existing ones[25]. For instance, given two MRs MR_1=(R_1, R_{f1}), MR_2=(R_2, R_{f2}), MR_1 is said to be composable with MR_2 if any follow-up test case of MR_1 can be used as the source test case of MR_2. The composition of two MRs is denoted as $MR_{1,2}$, and MR_1 and MR_2 are termed as the component MRs of $MR_{1,2}$. We refer readers to[27] on the discussion of the composability of MRs.

Recall the example in Section 2.1. With MR_1 and MR_2 as component MRs, we can derive MR $MR_{1,2}$ as follows, namely if the graph is permutated and the starting node and ending node are swapped, the shortest path will remain unchanged. Given a source test case $t = (G, a, b)$, a follow-up test case $t' = (G', b', a')$ can be derived based on $MR_{1,2}$ accordingly. With $MR_{1,2}$, we execute P with t and $t^{'}$ and check whether $|P(t)| = |P(t^{'})|$. Similarly, the violation of output relation in $MR_{1,2}$ indicates the detection of a fault.

Clearly, through the MR composition, one can derive a lot of composite MRs even with a small number of identified component MRs, which significantly reduces the efforts when a large number of diverse MRs are expected. Previous works also show that a set of diverse MRs can result in a higher fault detection effectiveness[56,57]. A plausible reason is that the more diverse MRs tend to cover more independent properties of a SUT and thus have a bigger chance of detecting faults that violates these properties in one way or another. Considering this advantage, MR composition will be leveraged in our proposed MR identification technique.

2.3 | Data Mutation (DM)

Data mutation was first proposed to solve the problem of generating structurally complex test cases[55]. The main idea is to generate new test cases by mutating existing ones according to some pre-defined rules, namely data mutation operators (DMOs). When DM is exercised, a small number of seed test cases should be first prepared, for instance, using a test case generation technique[16]. Secondly, a series of DMOs are designed based on the functional properties of the specification of SUT. Thirdly, mutated test cases are derived by applying DMOs to the seed test cases. In some situations, DMOs can be repeatedly applied to the derived mutated test cases to get more test cases. Fourthly, seed test cases and their mutated test cases are used to execute the SUT, during which various quantitative analysis methods can be used for the improvement of DMOs or the generation of additional seed test cases that are expected to have higher fault detection efficiency.

The unique feature of DM is that only a small number of seed test cases is required in advance, while a large number of mutated test cases can be generated by using DMOs. However, DM mainly concerns the test case generation, and validating the execution result of a mutated test case still requires the oracle, which is not addressed at all. Therefore, its applicability will be hindered if the test oracle problem exists in testing a SUT. In this paper, we leverage DM to direct a tester to effectively identify MRs and, at the same time, address the derivation of output relation, which DM has not addressed.

3 | APPROACH

In this section, we present a data mutation directed MR identification approach μMT, including its motivation, methodology, key issues, and illustration.

3.1 | Motivation

We first discuss the similarities between DM and MT for generating new test cases, as illustrated in Figure 2. They are summarized in the following two points.

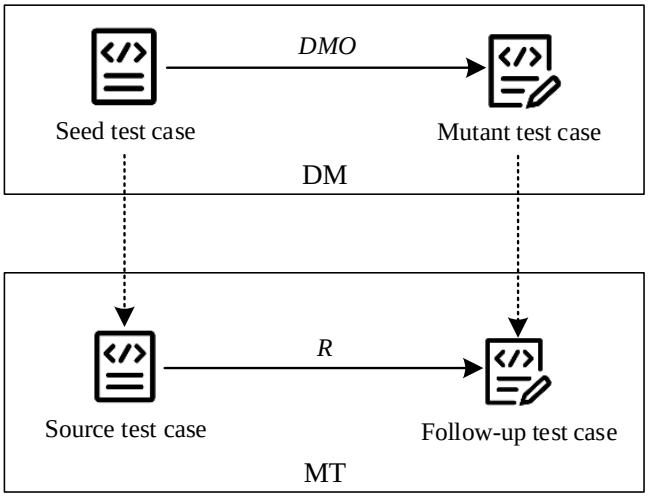


Figure 2 Commonalities between DM and MT

1) Both techniques can generate new test cases from existing test cases. DM generates new test cases (i.e., mutated test cases) from existing test cases according to user-defined DMOs. In contrast, MT generates new test cases (i.e., follow-up test cases) from existing ones (i.e., source test cases) based on the MRs.

2) Both techniques leverage similar steps to derive new test cases from existing ones. In DM, a DMO describes the rule to construct mutant test cases from a seed test case, while in MT, the input relation in an MR describes how a follow-up test case is related to a source test case.

One may view a DMO as a relation that describes the variation between two inputs, which can be an input relation of an MR, providing that the inputs quantified by the DMO are consistent with the necessary property of the SUT captured by the MR. In addition, the variation between inputs caused by a DMO can result in some predictable variation of outputs, which is also very similar to the output relation of the same MR that describes the necessary properties among multiple outputs. Accordingly, these similarities between DM and MT motivate us to derive MRs based on DM. Specifically, it can derive the MRs once we can figure out the possible variations of outputs that correspond to each of the DMOs according to the nature of SUT (i.e., the mapping rules). DMOs serve as guidance for systematically acquiring the input relation between inputs, and the mapping rules serve as clues to derive the output relation that corresponds to a given input relation. The input relation described by a DMO and the corresponding output relation form an MR for the SUT.

In addition, motivated by the MR composition technique, we propose to derive composite DMOs via the composition of DMOs, thereby obtaining more MRs and improving the cost-effectiveness of MT.

3.2 | The μMT Methodology

Figure 3 shows an overview of μMT. The main steps of μMT are described as follows.

1) *Input and output analysis*: μMT starts with analyzing the input and output parameters of SUT according to its specification, including the type of each parameter, the format of value, and the constraints on the value range.

2) *DMO selection*: A set of applicable data mutation operators (DMOs) are selected from a DMO set through an analysis of the input parameters. For instance, the tester can select the *swapping* for the triangle area calculation program.

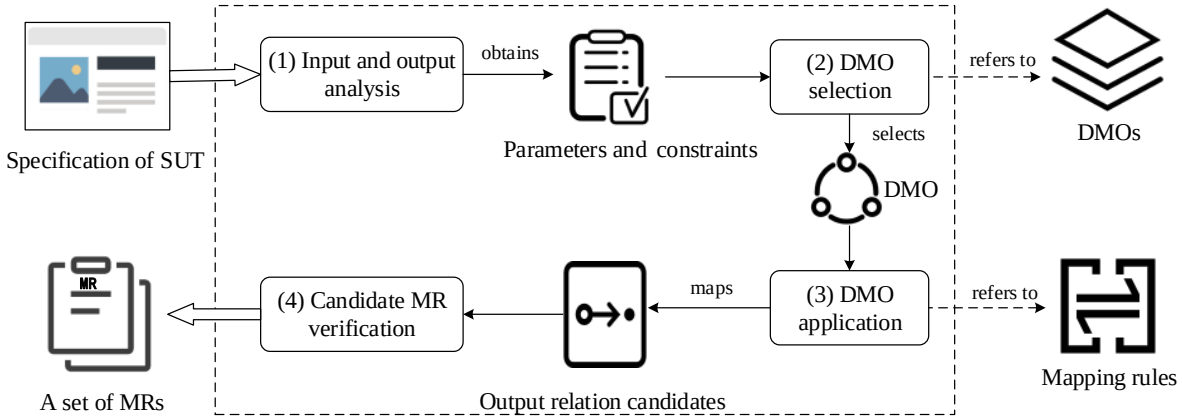


Figure 3 Basic principle of μMT

3) *DMO application*: For each selected DMO, an appropriate mapping rule from the part of all available rules is then selected to direct testers to focus on formulating an MR using the corresponding output relation template. We have summarized a set of potential output relations against each given DMO, which will be discussed in Section 3.3. A candidate MR is composed of the input relation indicated by the selected DMO and the output relation acquired through the selected mapping rule.

4) *Candidate MR verification*: The tester inspects the candidate MR to ensure that it conforms to some necessary property of SUT. If the candidate MR conforms to a necessary property of SUT, then it is regarded as a genuine MR; Otherwise, it is discarded.

μMT is different from either the original MT or DM in the following aspects: (i) The derivation of the input relation R in μMT is guided by DMOs, which is absent in the original MT; (ii) The derivation of the output relation R_f is guided by the mapping rules formulated based on the necessary properties of SUT, which is not considered in the original DM.

3.3 | DMOs and Mapping Rules

When the μMT methodology is exercised, DMOs and mapping rules are crucial for efficient MR identification. Accordingly, we propose seven individual DMOs and eight composite DMOs, respectively, while their mapping rules are provided in a template-style form, which facilitates the derivation of candidate MRs.

Table 1 summarizes the mapping rules with respect to individual DMOs, each of which describes the output relations that an individual DMO may apply. For each DMO except for $PeriT(x_i, T)$ and $Swap(x_i, x_j)$, two types of mapping rules are provided, i.e., equality and inequality output relations. When a DMO is selected to derive MRs, the mapping rule should be determined based on the properties of the SUT. In many scenarios, mapping rules with inequality output relations are recommended with a higher priority to derive MRs. Furthermore, deep domain knowledge and comprehension of the specification of SUT is helpful to derive more strict MRs using mapping rules with equality output relation.

We next elaborate on the usage of the mapping rules. Suppose that f is a monotonically increasing function with numeric inputs and outputs and $x_1, x_2, \ldots, x_n$ are the input parameters and $n > 1$.

1) $Swap(x_i, x_j)$: This DMO swaps the values of x_i and x_j, and the output relation is determined through the properties of f. If the swapping does not affect the output of f, then $Swap(x_i, x_j)$ is applicable and the candidate output relation is $f(x_1, \ldots, Swap(x_i, x_j), \ldots, x_n) = f(x_1, \ldots, x_n)$; Otherwise, $Swap(x_i, x_j)$ is not applicable for deriving MRs.

2) $Inc(x_i)$: This DMO adds the value of x_i by 1. The candidate output relation is $f(x_1, \ldots, inc(x_i), \ldots, x_n) > f(x_1, \ldots, x_n)$. If f is a linearly increasing function, an equality output relation can be derived, that is, $f(x_1, \ldots, Inc(x_i), \ldots, x_n) = f(x_1, \ldots, x_n) + g(x_1, \ldots, x_n)$, where $g(x_1, \ldots, x_n)$ is a function determined by the properties of SUT.

Table 1 Mapping rules for individual DMOs

ID	DMO	Candidate Output Relations	
		Type	Relation
1	$Swap(x_i, x_j)$	Equality	$f(x_1, \dots, Swap(x_i, x_j), \dots, x_n) = f(x_1, \dots, x_n)$
2	$Inc(x_i)$	Inequality	$f(x_1, \dots, Inc(x_i), \dots, x_n) > f(x_1, \dots, x_n)$ or $f(x_1, \dots, Inc(x_i), \dots, x_n) < f(x_1, \dots, x_n)$
		Equality	$f(x_1, \dots, Inc(x_i), \dots, x_n) = f(x_1, \dots, x_n) + g(x_1, \dots, x_n)$
3	$Dec(x_i)$	Inequality	$f(x_1, \dots, Dec(x_i), \dots, x_n) > f(x_1, \dots, x_n)$ or $f(x_1, \dots, Dec(x_i), \dots, x_n) < f(x_1, \dots, x_n)$
		Equality	$f(x_1, \dots, Dec(x_i), \dots, x_n) = f(x_1, \dots, x_n) - g(x_1, \dots, x_n)$
4	$Double(x_i)$	Inequality	$f(x_1, \dots, Double(x_i), \dots, x_n) > f(x_1, \dots, x_n)$ or $f(x_1, \dots, Double(x_i), \dots, x_n) < f(x_1, \dots, x_n)$
		Equality	$f(x_1, \dots, Double(x_i), \dots, x_n) = k * f(x_1, \dots, x_n) + g(x_1, \dots, x_n)$
5	$Half(x_i)$	Inequality	$f(x_1, \dots, Half(x_i), \dots, x_n) > f(x_1, \dots, x_n)$ or $f(x_1, \dots, Half(x_i), \dots, x_n) < f(x_1, \dots, x_n)$
		Equality	$f(x_1, \dots, Half(x_i), \dots, x_n) = k * f(x_1, \dots, x_n) - g(x_1, \dots, x_n)$
6	$PeriT(x_i, T)$	Equality	$f(x_1, \dots, PeriT(x_i, T), \dots, x_n) = f(x_1, \dots, x_n)$
7	$Neg(x_i)$	Inequality	$f(x_1, \dots, Neg(x_i), \dots, x_n) > (x_1, \dots, x_n)$ or $f(x_1, \dots, Neg(x_i), \dots, x_n) < (x_1, \dots, x_n)$
		Equality	$f(x_1, \dots, Neg(x_i), \dots, x_n) = f(x_1, \dots, x_n) - g(x_1, \dots, x_n)$

3) $Dec(x_i)$: This DMO decreases the value of x_i by 1. The candidate output relation is $f(x_1, \dots, dec(x_i), \dots, x_n) < f(x_1, \dots, x_n)$. In some situations, we can derive an equality output relation as $f(x_1, \dots, Dec(x_i), \dots, x_n) = f(x_1, \dots, x_n) - g(x_1, \dots, x_n)$.

4) $Double(x_i)$: This DMO doubles the value of x_i. The candidate output relation is $f(x_1, \dots, double(x_i), \dots, x_n) > f(x_1, \dots, x_n)$. In some situations, we can derive an equality output relation as $f(x_1, \dots, Double(x_i), \dots, x_n) = f(x_1, \dots, x_n) + g(x_1, \dots, x_n)$, where k is the proportion coefficient.

5) $Half(x_i)$: This DMO halves the value of x_i. The candidate output relation is $f(x_1, \dots, half(x_i), \dots, x_n) < f(x_1, \dots, x_n)$. In some situations, the candidate output relation can be mapped as $f(x_1, \dots, Half(x_i), \dots, x_n) = k * f(x_1, \dots, x_n) - g(x_1, \dots, x_n)$.

6) $PeriT(x_i, T)$: This DMO adds T to the value of x_i $(0 < i \leq n)$, namely $PeriT(x_i, T) = x_i + T$. If f is a periodic function with a period of T, then the candidate output relation is $f(x_1, \dots, PeriT(x_i, T), \dots, x_n) = f(x_1, \dots, x_n)$; otherwise, this DMO is not applicable.

7) $Neg(x_i)$: This DMO negates the value of x_i. The candidate output relation is $f(x_1, \dots, Neg(x_i), \dots, x_n) < f(x_1, \dots, x_n)$. In some situations, the output relation can be mapped as $f(x_1, \dots, Neg(x_i), \dots, x_n) = f(x_1, \dots, x_n) - g(x_1, \dots, x_n)$.

We next present the mapping rules for composite DMOs. Given two DMOs A and B, A is said to be composable to B if the outcome of applying A to any inputs of SUT is applicable to B. Accordingly, among seven DMOs in Table 1, after our manual analysis of all pairs, we find that four of them can be composed, and eight composite DMOs are thus provided, as shown in Table 2.

Recall the aforementioned monotonically increasing function f. We illustrate two of the eight mapping rules for composite DMOs as follows:

1) $Inc\_Double(x_i)$: The value of x_i is first increased by one and then doubled, that is, $Inc\_Double(x_i) = Double(Inc(x_i)) = (x_i + 1) * 2$. Accordingly, the candidate output relation is $f(x_1, \dots, Inc\_Double, (x_i), \dots, x_n) > f(x_1, \dots, x_n)$. If f is a linearly increasing function, an equality output relation can be derived as $f(x_1, \dots, Inc\_Double\,(x_i), \dots, x_n) = k * f(x_1, \dots, x_n) + g(x_1, \dots, x_n)$.

2) $Half\_Dec(x_i)$: The value of x_i is first halved and then decreased by 1, that is, $Half\_Dec(x_i) = Dec(Half(x_i)) = x_i/2 - 1$. Accordingly, the candidate output relation is $f(x_1, \dots, Half\_Dec(x_i), \dots, x_n) < f(x_1, \dots, x_n)$. If f is a linearly increasing function, an equality output relation can be derived as $f(x_1, \dots, Half\_Dec(x_i), \dots, x_n) = k * f(x_1, \dots, x_n) - g(x_1, \dots, x_n)$.

Table 2 Mapping rules for composite DMOs

ID	DMO	Candidate Output Relations
1	$Inc\_Double(x_i)$	$f(x_1,\dots,Inc\_Double(x_i),\dots,x_n) > f(x_1,\dots,x_n)$ or $f(x_1,\dots,Inc\_Double(x_i),\dots,x_n) < f(x_1,\dots,x_n)$ or $f(x_1,\dots,Inc\_double(x_i),\dots,x_n) = k * f(x_1,\dots,x_n) + g(x_1,\dots,x_n)$
2	$Double\_Inc(x_i)$	$f(x_1,\dots,Inc\_Double(x_i),\dots,x_n) > f(x_1,\dots,x_n)$ or $f(x_1,\dots,Inc\_Double(x_i),\dots,x_n) < f(x_1,\dots,x_n)$ or $f(x_1,\dots,Inc\_Double(x_i),\dots,x_n) = k * f(x_1,\dots,x_n) + g(x_1,\dots,x_n)$
3	$Dec\_Half(x_i)$	$f(x_1,\dots,Dec\_Half(x_i),\dots,x_n) > f(x_1,\dots,x_n)$ or $f(x_1,\dots,Dec\_Half(x_i),\dots,x_n) < f(x_1,\dots,x_n)$ or $f(x_1,\dots,Dec\_Half(x_i),\dots,x_n) = k * f(x_1,\dots,x_n) - g(x_1,\dots,x_n)$
4	$Half\_Dec(x_i)$	$f(x_1,\dots,Half\_Dec(x_i),\dots,x_n) > f(x_1,\dots,x_n)$ or $f(x_1,\dots,Half\_Dec(x_i),\dots,x_n) < f(x_1,\dots,x_n)$ or $f(x_1,\dots,Half\_Dec(x_i),\dots,x_n) = k * f(x_1,\dots,x_n) - g(x_1,\dots,x_n)$
5	$Inc\_Half(x_i)$	$f(x_1,\dots,Inc\_Half(x_i),\dots,x_n) > f(x_1,\dots,x_n)$ or $f(x_1,\dots,Inc\_Half(x_i),\dots,x_n) < f(x_1,\dots,x_n)$ or $f(x_1,\dots,Inc\_Half(x_i),\dots,x_n) = k * f(x_1,\dots,x_n) + g(x_1,\dots,x_n)$
6	$Half\_Inc(x_i)$	$f(x_1,\dots,Half\_Inc(x_i),\dots,x_n) > f(x_1,\dots,x_n)$ or $f(x_1,\dots,Half\_Inc(x_i),\dots,x_n) < f(x_1,\dots,x_n)$ or $f(x_1,\dots,Half\_Inc(x_i),\dots,x_n) = k * f(x_1,\dots,x_n) + g(x_1,\dots,x_n)$
7	$Dec\_Double(x_i)$	$f(x_1,\dots,Dec\_Double(x_i),\dots,x_n) > f(x_1,\dots,x_n)$ or $f(x_1,\dots,Dec\_Double(x_i),\dots,x_n) < f(x_1,\dots,x_n)$ or $f(x_1,\dots,Dec\_double(x_i),\dots,x_n) = k * f(x_1,\dots,x_n) - g(x_1,\dots,x_n)$
8	$Double\_Dec(x_i)$	$f(x_1,\dots,Double\_Dec(x_i),\dots,x_n) > f(x_1,\dots,x_n)$ or $f(x_1,\dots,Double\_Dec(x_i),\dots,x_n) < f(x_1,\dots,x_n)$ or $f(x_1,\dots,Double\_Dec(x_i),\dots,x_n) = k * f(x_1,\dots,x_n) - g(x_1,\dots,x_n)$

3.4 | Illustration

We use an example to demonstrate how μMT works. Suppose that P is an implementation of $f(a,b)$ that calculates the area of a rectangle where a and b denote the length and width of the rectangle, respectively. The MR derivation is described as follows.

Table 3 Derived MRs for the rectangle area calculation program

ID	DMOs	Output Relations
1	$Swap(a,b)$	$f(Swap(a,b)) = f(a,b)$
2	$Inc(a)$	$f(Inc(a),b) > f(a,b)$ or $f(Inc(a),b) = f(a,b) + b$
3	$Dec(a)$	$f(Dec(a),b) < f(a,b)$ or $f(Dec(a),b) = f(a,b) - b$
4	$Double(a)$	$f(Double(a),b) > f(a,b)$ or $f(Double(a),b) = 2 * f(a,b)$
5	$Half(a)$	$f(Half(a),b) < f(a,b)$ or $f(Half(a),b) = 0.5 * f(a,b)$
6	$Inc\_Double(a)$	$f(Inc\_Double(a),b) > f(a,b)$ or $f(Inc\_Double(a),b) = 2 * f(a,b) + 2 * b$
7	$Double\_Inc(a)$	$f(Double\_Inc(a),b) > f(a,b)$ or $f(Double\_Inc(a),b) = 2 * f(a,b) + b$
8	$Dec\_Half(a)$	$f(Dec\_Half(a),b) < f(a,b)$ or $f(Dec\_Half(a),b) = 0.5 * f(a,b) - 0.5 * b$
9	$Half\_Dec(a)$	$f(Half\_Dec(a),b) < f(a,b)$ or $f(Half\_Dec(a),b) = 0.5 * f(a,b) - b$
10	$Inc\_Half(a)$	$f(Inc\_Half(a),b) < f(a,b)$ or $f(Inc\_Half(a),b) = 0.5 * f(a,b) + 0.5 * b$
11	$Half\_Inc(a)$	$f(Half\_Inc(a),b) < f(a,b)$ or $f(Half\_Inc(a),b) = 0.5 * f(a,b) + b$
12	$Dec\_Double(a)$	$f(Dec\_Double(a),b) > f(a,b)$ or $f(Dec\_Double(a),b) = 2 * f(a,b) - 2 * b$
13	$Double\_Dec(a)$	$f(Double\_Dec(a),b) > f(a,b)$ or $f(Double\_Dec(a),b) = 2 * f(a,b) - b$

1) By analyzing the specification of $f(a,b)$, we get the input parameters a, b, and $a > 0, b > 0$.

2) On the basis of input analysis, five of seven mapping rules for individual DMOs in Table 1 (i.e., Rules 1, 2, 3, 4, and 5) and all eight mapping rules for composite DMOs in Table 2 are applicable. Rule 6 (for $PeriT(x_i, T)$) in Table 1 is not applicable because the rectangle area calculation function should not be periodic; Similarly, Rule 7 (for $Neg(x_i)$) in Table 1 is not applicable, because the length of a side of the rectangle cannot be negative.

3) By expert knowledge of geometry, we can figure out that $f(a, b)$ is a monotonically increasing function; that is, the area of a rectangle will increase (or decrease) as the length or width of the rectangle increases (or decreases) while the other is kept the same. As an illustration, consider Rule 3 (for $Dec(x_i)$) in Table 1. Keep in mind the properties of $f(a, b)$, we can infer that the area of a rectangle will decrease when a decreases, thus $f(Dec(a), b) < f(a, b)$ is a candidate output relation according to inequality mapping rules [1]. Similarly, the above process can be repeated for other applicable rules, and the resulting candidate MRs are summarized in Table 3.

4) Finally, we manually inspect the derived MRs. When Rule 3 is considered since $Dec(a)$ decreases a by 1, the constraint "$a > 1$" is necessary for this MR; otherwise, after the mutation of $Dec(a)$, the length of the side a' in follow-up test case becomes negative, which violates the specification of $f(a, b)$. We can identify a set of genuine MRs for P through refinement.

4 | PROTOTYPE TOOL

In our previous work[58], MT4WS was developed to automate the MT of Web services as much as possible. The frontend was implemented in Java, while the backend was implemented in JSP, and the MySQL database was used. Figure 4 shows the architecture of MT4WS. MT4WS first parses the WSDL document of the Web service under test to obtain the operations and their corresponding input and output parameters. With the signatures of operation under test, the tester may define MRs manually or import MRs from an MRDL file with the aid of the tool. For the derived MRs, MT4WS can randomly generate a specified number of source test cases and derive follow-up test cases accordingly. Then MT4WS executes in batch the Web service under test with the source test cases and follow-up test cases and intercepts their corresponding outputs. Finally, a test report will be produced indicating the result of all the executed tests. Various configurations are supported to make the MT process as flexible as possible.

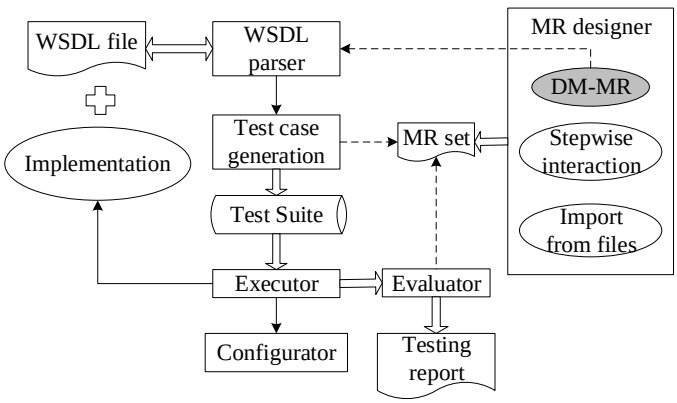


Figure 4 Architecture of MT4WS

In this work, we further extended MT4WS to support μMT. The original MT4WS has six components as shown in Figure 4, namely *MR designer*, *WSDL parser*, *Test case generator*, *Configurator*, *Executor*, and *Evaluator*. Among these components, *MR designer* is responsible for designing MRs for Web services under test. Two ways of MR derivation are supported, namely

[1] For simplicity, we take the input parameter a for illustration. Similar mutation can also be applied to the input parameter b.

Stepwise Interaction, which derives MRs manually in an interactive manner and *Import from files*, which derives MRs by importing from an XML file in a batch manner. To support our proposed μMT, we have extended the *MR designer* component of MT4WS with a *DM-MR* module. Figure 5 shows a snapshot of using μMT to derive MRs, which is further detailed as follows:

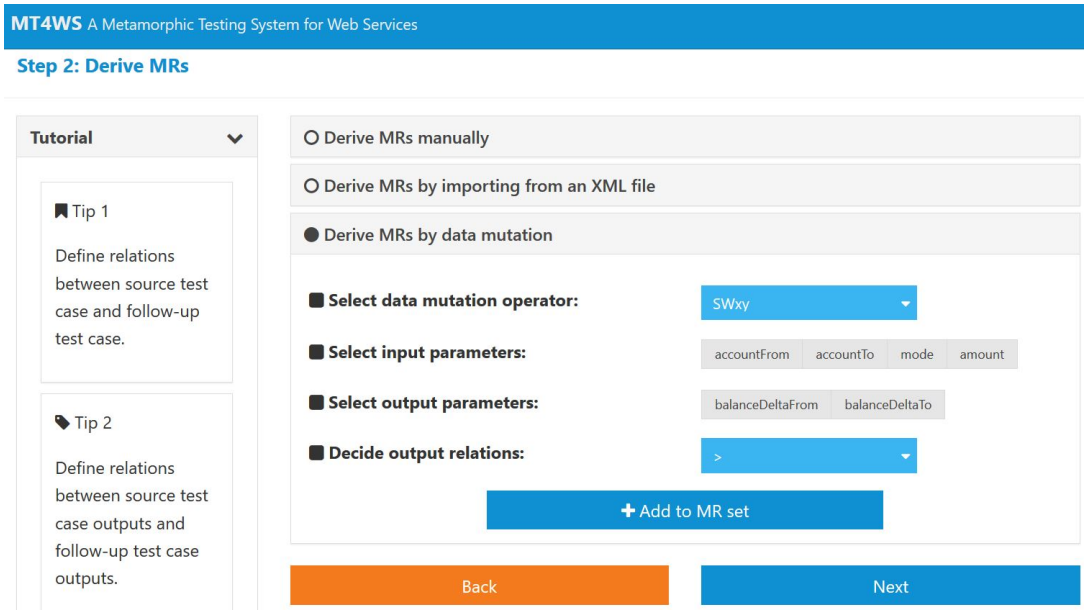


Figure 5 Snapshot of deriving MRs using μMT

1. The "Derive MRs by data mutation" option is first selected, and the system will initialize a set of applicable DMO sets, from which the tester then selects one for the MR identification.
2. The system then shows the input and output parameters, and the tester decides the relation of the outputs according to the properties of the operation under test. The system generates MR from the options the tester selects per some mapping rule. All mapping rules for individual DMOs in Table 1 and composite DMOs in Table 2 are supported.
3. Before storing the candidate MR, the system will verify the DMO selected by the tester and the desirable number of input parameters according to the mapping rules. If there are any violations, a warning message will be provided. For example, if $Swap(x_i, x_j)$ is selected, two input parameters and one output parameter must be selected.
4. If the tester confirms the derived candidate MR, then the system converts and exports the identified MRs into an MRDL-style file. MRDL provides an XML-based formal expression of MRs. Interested readers may refer to[58] for more details.

Besides the MR identification, the tool is also able to automatically generate follow-up test cases based on the input relations of the identified MRs. For each selected DMO, the source test cases are used as the seed test cases of DM, and the DMO is used to generate mutant test cases. Only valid mutated test cases are selected into follow-up test cases. A mutant test case is validated whether it satisfies the given constraints over the input domain of SUT. For example, assuming that a source test case $< 0.5, 1, 1.2 >$ is given, which denotes a triangle with three side lengths of 0.5, 1, and 1.2, and if $Dec(x_i)$ is selected, three mutant test cases, $< -0.5, 1, 1.2 >$, $< 0.5, 0, 1.2 >$, and $< 0.5, 1, 0.2 >$, are generated provided that the DMO changes only one parameter once. Moreover, some mutant test cases violate the constraint for an invalid triangle. They are regarded as invalid follow-up test cases. Finally, the same process of MT4WS applies to the execution of MTG and verification of test results.

5 | SETTINGS OF EMPIRICAL STUDY

This section describes the settings of our empirical study to evaluate the MR identification effectiveness of μMT and the performance of identified MRs using μMT.

5.1 | Research Questions

Our empirical study aims at answering the following research questions:

RQ1: How effective is μMT in MR identification?

We apply μMT to a set of diverse subject programs and examine whether MRs can be effectively identified. We compare the number of trials required to identify a given size MR set using μMT and the random technique.

RQ2: How effective are MRs identified using μMT in fault detection?

We measure the fault detection effectiveness in terms of mutation scores of MRs identified using μMT. The performance of MT is related to not only the used MRs but also their associated test cases. Accordingly, together with statement coverage, as an indicator of testing adequacy of associated test cases, mutation scores can comprehensively evaluate the fault detection effectiveness of μMT.

RQ3: How effective is μMT compared with related MR identification techniques based on data mutation?

To answer this question, mutational MT[22] is considered as a baseline technique because it shares the same starting point and restriction with μMT, and thus the comparison is fair. That is, both of them are developed for numeric programs and leverage data mutation to identify MRs. We compare the number and mutation score of MRs identified using μMT with that of mutation MT.

RQ4: Does the type of mapping rules for the same DMO affect the fault detection effectiveness of MRs identified using μMT?

We identify two groups of MRs for the same DMO using equality and inequality output mapping rules and then compare the fault detection effectiveness of their identified MRs.

RQ5: Does the type of DMOs affect the fault detection effectiveness of MRs identified using μMT?

We decompose this question into the following two sub-questions:

RQ5.1: Does the fault detection effectiveness of derived MRs using μMT vary with individual DMO?

RQ5.2: Does the fault detection effectiveness of derived MRs using composite DMOs correlate with that of individual DMOs?

Table 4 Summary of subject programs and their testing information

Subject program	LOC	Faulty versions	Source of faulty versions	Number of MTGs
ATM	263	48	Using MuJava with mutation operators: AOIS, AORB COI, LOI, ROR, AOIU, ODL, SDL, VDL	10, 50, 100
BillCal	113	162	Using MuJava with mutation operators: AOIS, AORB COI, LOI, ROR, AOIU, ODL, SDL, VDL	10, 50, 100
BaggBill	215	67	Using MuJava with mutation operators: AOIS, AORB COI, LOI, ROR, AOIU, ODL, SDL, VDL	10, 50, 100
ParkingFee	266	810	Using MuJava with mutation operators: AOIS, AORB COI, LOI, ROR, AOIU, ODL, SDL, VDL	10, 50, 100
NumberUtil	1438	4	Real Bugs from Apache Common Lang in Defects4J: Bug IDs: 1, 16, 24, 58	10, 50, 100
TaxBill	2150	1565	Using MuJava with mutation operators: AOIS, AORB COI, LOI, ROR, AOIU, ODL, SDL, VDL	10, 50, 100

5.2 | Subject programs

Our experiments included six subject programs manifesting diverse application domains with different sources. More specifically,

- Automatic Teller Machine Transfer System (ATM)[59] offers a set of features such as withdrawal, deposit, transfer, and query. Among them, the *transfer* operation was selected, which calculates the commission fee based on the China Agriculture Bank's policy.
- China Unicom Billing System (BillCal)[1] calculates the customer's monthly bill according to China Unicom's billing policy. The *PhoneBillCalculation* operation accepts the input data, including call time, free base call time, call rate, flow, free base flow, flow rate, and monthly base fee. For two types of monthly packages (A or B), varying amounts of free base call time, free base flow, call rate, and flow rate are allocated.
- Airline Baggage Check-in Billing System (BaggBill)[1] is an aviation baggage billing service based on Air China's baggage billing standard. For the class of air tickets, passengers are allowed to carry certain limited pieces and weights of luggage, and overweight or additional luggage has to be charged (implemented by the *feeCalculation* operation) based on the billing policy.
- Parking Billing System (ParkingFee)[23] calculates the parking fee based on the driver's vehicle type, car type, parking date, discount coupon, and parking time, with reference to the billing criteria of a car park (implemented by the *parkingFeeWithoutDiscount* operation).
- NumberUtil[60] is a data type conversion program in the Apache Common Lang library for Java. The program converts a number of the String type into a decimal number of the Number type based on a series of conversion rules. The *createNumber* operation provides the primary conversion function, which accepts a number in the string form and outputs that number in the Number type.
- Commodity Tax Billing System (TaxBill) [2] calculates the total amount of sale taxes paid for the purchased product. The *gettax* operation accepts the list of goods with amount, price, and rate of tax and outputs the total amount of taxes.

All subject programs are written in Java. The size of each subject program, measured by lines of code (LOC), is given in the second column of Table 4. In this study, we have not included non-numeric programs for evaluation since DMOs used by μMT are mainly related to numbers. The inclusion of non-numeric programs for evaluation will contribute to a more comprehensive evaluation of μMT. However, this involves the design of new DMOs and corresponding output relations, which will significantly expand the scope of this study and deserve a separate study, and is left as future work.

5.3 | Faulty versions of subject programs

Our empirical study used both artificial faults and real-life faults. For NumberUtil, we collected its real-life faults from an open-source project[60], and four faulty versions were thereby used. For the other five subject programs, MuJava[61] was employed to generate mutants according to different types of mutation operators, and each mutant contained only one artificial fault.

We first conducted a preliminary test using a test suite with 5,000 test cases to identify the equivalent mutants that are functionally equivalent to the original program. Then, we manually examined those alive mutants and removed the equivalent ones before the evaluation of techniques. Our experiments identified 340, 50, 58, 327, and 414 equivalent mutants for ATM, BillCal, BaggBill, ParkingFee, and TaxBill. As a result, 48, 162, 67, 810, and 1,565 non-equivalent mutants were included in the evaluations on ATM, BillCal, BaggBill, ParkingFee, and TaxBill, respectively. The third and fourth columns of Table 4 summarize the basic information of these faulty versions.

5.4 | Source test case generation

In our experiments, random testing was used to generate source test cases according to the specification of SUT. In order to study the impact of the size of source test suites on the fault detection effectiveness of MRs. We conducted a preliminary trial.

[2]https://github.com/elainechan/sales-tax-calculator

We observed that the fault detection effectiveness was almost the same when the size of source test suites was larger than 100 [3]. Specifically, we further included effect size to measure the magnitude of the mutation score differences between the test suites of 100 (dataset 1) and 200 (dataset 2). Cohen's d[62] is a popularly adopted effect size and thus was used in our experiments. Cohen's d value is defined as follows:

$$Cohen's\ d = \frac{M_2 - M_1}{\sqrt{\frac{SD_1^2 + SD_2^2}{2}}} \tag{2}$$

where M_1 and M_2 denote the mean values of dataset 1 and dataset 2, respectively, and SD_1 and SD_2 denote the standard deviations of dataset 1 and dataset 2, respectively. Cohen's d values were summarized in Table 5. Clearly, the mutation score

Table 5 Cohen's d Values for Test Suite Sizes of 100 and 200

Subject Program	Cohen's d
ATM	0.00
BillCal	0.54
BaggBill	0.07
ParkingFee	0.54
NumberUtil	0.00
TaxBill	0.40

differences between the two test suite sizes are small for ATM, BaggBill, and NumberUtil (i.e., Cohen's d $\leqslant$ 0.2), while medium for BillCal, ParkingFee, and TaxBill (i.e., Cohen's d $\approx$ 0.5). Cohen's d analysis indicated that the increase in the test suite size would not significantly impact mutation scores. Accordingly, we set the maximum size of test suites to 100. We generated three source test suites for each subject program by randomly generating 10, 50, and 100 source test cases for each MR, as shown in the fifth column of Table 4.

5.5 | Metrics

In our experiments, the fault detection effectiveness of identified MRs is measured in terms of fault detection capability and testing adequacy. Mutation score (MS) was used to measure the fault detection effectiveness by the ratio of the number of killed mutants to that of the non-equivalent mutants. In our experiments, a faulty version corresponds to a mutant or a real fault which is said to be killed/detected if an MR derived using our approach is violated. The mutation score is defined as follows:

$$MS(p, ts) = \frac{N_k}{N_m - N_e}, \tag{3}$$

where p and ts denote the program under test and the used test suite, respectively; N_k, N_m, and N_e denote the number of killed mutants, the total number of mutants, and the number of equivalent mutants, respectively. A higher $MS(p, ts)$ indicates better fault detection capability.

Statement coverage (SC) was used to measure the fault detection effectiveness by the extent to which source code in the subject program is executed, which provides evidence for the testing adequacy evaluation. The statement coverage is defined as follows:

$$SC(p, ts) = \frac{ST_{executed}}{ST_{total}}, \tag{4}$$

where p denotes program under test, ts denotes the used test suite, $ST_{executed}$ is the number of executed statements and ST_{total} denotes the total number of executable statements. The higher $SC(p, ts)$ is, the more testing adequacy is achieved. In our experiments, we leveraged a code coverage analysis tool to collect and analyze the statement coverage information of each test case during test execution. The tool is provided by IntelliJ IDEA [4].

[3] Details of the identified MRs and mutation score for a test suite of 200 test cases are available at https://github.com/MuMT-USTB .
[4] https://www.jetbrains.com/help/idea/code-coverage.html

6 | EXPERIMENTAL RESULTS AND ANALYSIS

6.1 | Answer to RQ1 (MR identification effectiveness of μMT)

Following the methodology presented in Section 3.2, we applied μMT to the subject programs, as illustrated in Section 3.4. Table 6 provided a summary of the derived MRs for all subject programs using the different DMOs. In our experiments, μMT successfully derived 12, 96, 60, 52, 156, and 144 MRs for ATM, BillCal, BaggBill, ParkingFee, NumberUtil, and TaxBill, respectively[3].

The output relations of the mapping rules with respect to an applicable DMO could be either equality or inequality. Since equality MRs are more constrained than inequality ones, their fault detection effectiveness is expected to be higher, and thus equality MRs were identified for the subsequent evaluation.

Table 6 Summary of MRs derived using μMT

DMO	Subject Program					
	ATM	BillCal	BaggBill	ParkingFee	NumberUtil	TaxBill
$Inc(x_i)$	1	8	5	6	13	12
$Dec(x_i)$	1	8	5	6	13	12
$Double(x_i)$	1	8	5	4	13	12
$Half(x_i)$	1	8	5	4	13	12
$Inc\_Double(x_i)$	1	8	5	4	13	12
$Double\_Inc(x_i)$	1	8	5	4	13	12
$Dec\_Half(x_i)$	1	8	5	4	13	12
$Half\_Dec(x_i)$	1	8	5	4	13	12
$Inc\_Half(x_i)$	1	8	5	4	13	12
$Half\_Inc(x_i)$	1	8	5	4	13	12
$Dec\_Double(x_i)$	1	8	5	4	13	12
$Double\_Dec(x_i)$	1	8	5	4	13	12
Total	12	96	60	52	156	144

For each subject program, twelve DMOs are applicable for identifying MRs, including four individual DMOs ($Inc(x_i)$, $Dec(x_i)$, $Double(x_i)$, and $Half(x_i)$) and eight composite DMOs ($Inc\_Double(x_i)$, $Double\_Inc(x_i)$, $Dec\_Half(x_i)$, $Half\_Dec(x_i)$, $Dec\_Double(x_i)$, $Double\_Dec(x_i)$, $Inc\_Half(x_i)$, and $Half\_Inc(x_i)$). Note that two individual DMOs, i.e., $PeriT(x_i, T)$ and $Neg(x_i)$, were not applicable in our experiments since all subject programs have no periodic property and only accept positive inputs. However, this does not necessarily mean these two DMOs are not useful. For instance, assume the program under test implements the sine function $sin(x)$. Thus, $PeriT(x_i, T)$ can be used to derive an MR like $sin(x_i + 2\pi) = sin(x_i)$ where T is 2π; Similarly, $Neg(x_i)$ can be used to derive an MR like $sin(-x_i) = -sin(x_i)$.

To demonstrate the effectiveness of μMT, random selection was used as a baseline for comparison. When identifying MRs using the random technique, the twelve DMOs mentioned above are used as the candidate DMOs, while the candidate output relations are *greater than* ($>$), *less than* ($<$), *equal to* ($=$), *greater than or equal to* ($\geq$), and *less than or equal to* ($\leq$). For each subject program, we counted the number of trials when μMT was used; the random technique first randomly selected a DMO from the candidate DMOs and then randomly tried an output relation from the candidate output relations, and the number of trials was recorded when the same size set of genuine MRs was identified. As suggested by[63] that at least thirty observations are required to ensure the statistical significance of the results, we repeated the experiments thirty times.

Table 7 summarizes the number of trials for both techniques to identify the same size set of MRs, where *AVG* represents the mean value of thirty times and *SD* represents the standard deviation. For example, for the subject program ATM, μMT just tried twelve times to identify twelve MRs, each being identified with a different DMO. In comparison, the random technique was tried on average 92 times to identify the same size set of MRs. From Table 7, we observe that the random technique takes much more trials than μMT to identify the same size set of MRs, which further indicates the effectiveness of μMT.

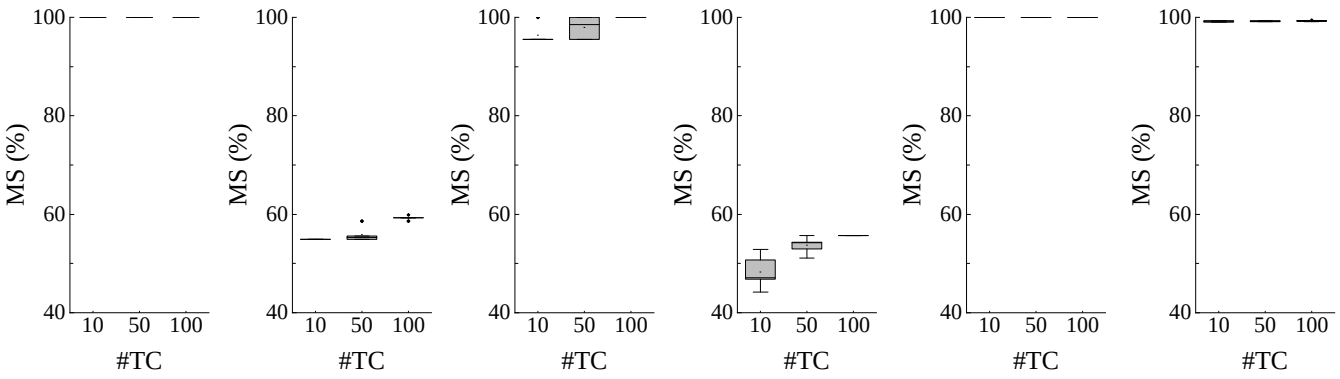
Table 7 Number of trials used for generating the same number of MR sets

techniques	Subject Program											
	ATM		BillCal		BaggBill		ParkingFee		NumberUtil		TaxBill	
	AVG	SD	AVG	SD	AVG	SD	AVG	SD	AVG	SD	AVG	SD
μMT	12	0	96	0	60	0	52	0	156	0	144	0
Random	92	32	734	80	451	73	561	97	1243	111	1133	111

Based on the above evaluation, we conclude that μMT can effectively direct testers to identify MRs for the benchmark programs.

6.2 | Answer to RQ2 (Fault detection effectiveness of MRs identified using μMT)

For each program, we collected the mutation score and the statement coverage for each equality MR with respect to three sizes of test suites. The experiments were repeated thirty times. Figure 6 shows the mutation score results for each program with respect to three test suites. In each boxplot, the upper and lower bounds of the box represent the third and first quartiles of the metric, respectively; the middle line represents the median value; the upper and lower whiskers denote the largest and smallest data within the range of $\pm 1.5 \times IQR$ (where IQR is the interquartile range), respectively; outliers beyond the IQR are denoted with hollow circles; and the solid circle represents the mean value of the metric. The '#TC' axis denotes the size of test suites, while the 'MS (%)' axis denotes mutation scores. From Figure 6, we have the following observations:



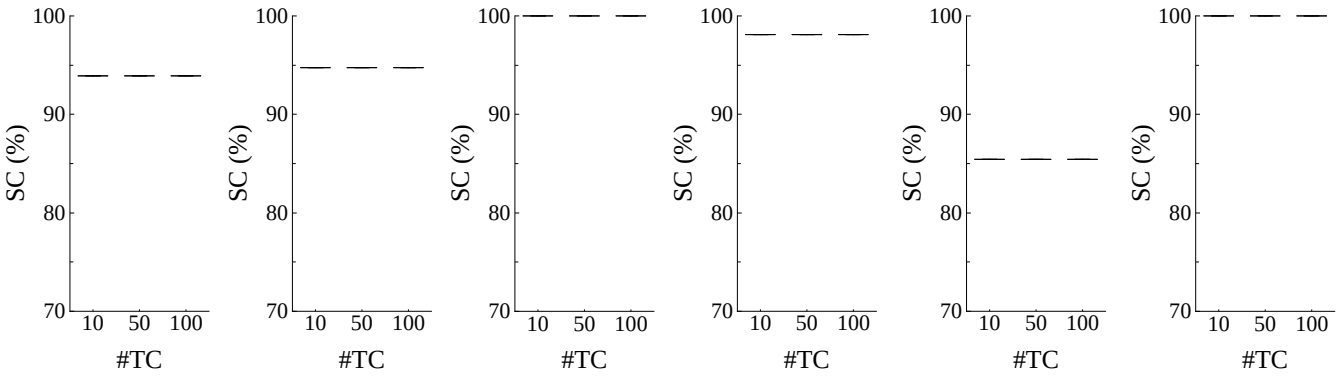
(a) ATM (b) BillCal (c) BaggBill (d) ParkingFee (e) NumberUtil (f) TaxBill

Figure 6 Mutation scores of derived equality MRs

- MRs identified using μMT achieved high mutation scores, demonstrating a good fault detection capability. Mutations scores of ATM (100%), BaggBill (98.36%), NumberUtil (100%), and TaxBill (99.65%) are almost 100%, when the size of test suites was 100, indicating that μMT were able to detect almost all faults. However, mutation scores of BillCal (57.14%) and ParkingFee (55.50%) were a bit low, around 60%.
- The mutation scores showed small fluctuations with varying test suite sizes for four subject programs. Specifically, the test suite size slightly affected the fault detection capability for BillCal, BaggBill, ParkingFee, and TaxBill, but there is no statistical difference in impact for ATM and NumberUtil.

Figure 7 shows the statement coverage results for each program with respect to three test suites, where the 'SC (%)' axis denotes statement coverage. Various studies have shown that higher code coverage improves fault detection capabilities and the effectiveness of software reliability assessment[64]. In Figure 7, we have the following observations:

- The statement coverage ranged from 85.42% to 100%, with a mean of 95.36% for all subject programs. This indicates that the test suites generated by μMT achieved a high statement coverage. By examining uncovered statements, we found that most of the uncovered statements were related to exception handling. This is because μMT only generated valid inputs according to the derived MRs, and there was no chance to trigger those exception-handling statements accordingly.
- Unlike the fluctuations of mutation scores with varying test suites, the statement coverage was stable across all subject programs. This further indicated that in the context of MT, there were no close correlations between statement coverage and fault detection.



(a) ATM (b) BillCal (c) BaggBill (d) ParkingFee (e) NumberUtil (f) TaxBill

Figure 7 Statement coverage of derived equality MRs

In summary, the MRs identified using μMT had a good fault detection capability in terms of achieving nearly 100% mutation scores for four out of six subject programs. Moreover, test cases generated according to the identified MRs consistently delivered a high statement coverage ranging from 85% to 100%, indicating high testing adequacy.

6.3 | Answer to RQ3 (Effectiveness comparison with baseline techniques)

In our experiments, mutational MT[22] was chosen as a baseline technique as it shared a similar principle with μMT, i.e., leveraging data mutation operators to identify MRs. Naturally, it is interesting to compare the MR identification effectiveness of both techniques and the fault detection effectiveness of their identified MRs. To make a fair comparison, for each program, the test suites with the same size (i.e., 100 test cases) were used for MRs identified using both mutational MT and μMT. Note that when mutational MT was used to identify MRs, only those DMOs mentioned in[22] were used.

Table 8 Comparison of numbers of identified MRs using Mutational MT and μMT

Technique	Subject Program					
	ATM	BillCal	BaggBill	ParkingFee	NumberUtil	TaxBill
Mutational MT	4	32	20	24	47	48
μMT	12	96	60	52	156	144

Table 8 summarizes the MR identification effectiveness in terms of the number of MRs identified using mutational MT and μMT. From Table 8, we observed the following:

- μMT identified much more MRs than mutational MT across six subject programs. This was mainly because that μMT proposed more DMOs and composite DMOs for MR identification than mutational MT. Specifically, twelve DMOs proposed by μMT were used as shown in Table 6, while only four DMOs proposed by mutational MT were used, namely IVP (Increasing the value of a parameter by 1, corresponding to $Inc(x_i)$ in μMT), DVP (Decreasing the value of a parameter by 1, corresponding to $Dec(x_i)$ in μMT), SPL (Setting the value of a parameter to a very large number, say 1,000,000), and SPZ (Setting the value of a parameter to 0).
- The numbers of identified MRs using both techniques significantly varied with subject programs. For ATM, the difference is only eight, while the differences are 109 and 96 for NumberUtil and TaxBill, respectively.

Table 9 shows the mutation score results for each program achieved by mutational MT and μMT. We observed that the mutation scores of MRs identified using μMT were much higher than that of mutational MT, except that both achieved a mutation score of 100% for NumberUtil. Surprisingly, μMT achieved a mutation score of 100% for ATM, while mutational MT only achieved 41.67%. Similar mutation score gaps were observed for ParkingFee and TaxBill.

Table 9 Comparison of mutation scores of identified MRs using Mutational MT and μMT

Technique	Subject Program					
	ATM	BillCal	BaggBill	ParkingFee	NumberUtil	TaxBill
Mutational MT	41.67%	49.38%	85.07%	19.01%	100.00%	41.02%
μMT	100.00%	57.14%	98.36%	55.50%	100.00%	99.65%

The overall experimental results showed that μMT was able to identify much more MRs and detect more faults than mutational MT under the circumstance where the same sizes of test suites were used. This further demonstrated that μMT had a better MR identification effectiveness, and its identified MRs had a better fault detection capability.

6.4 | Answer to RQ4 (Impact of mapping rules on fault detection effectiveness of identified MRs)

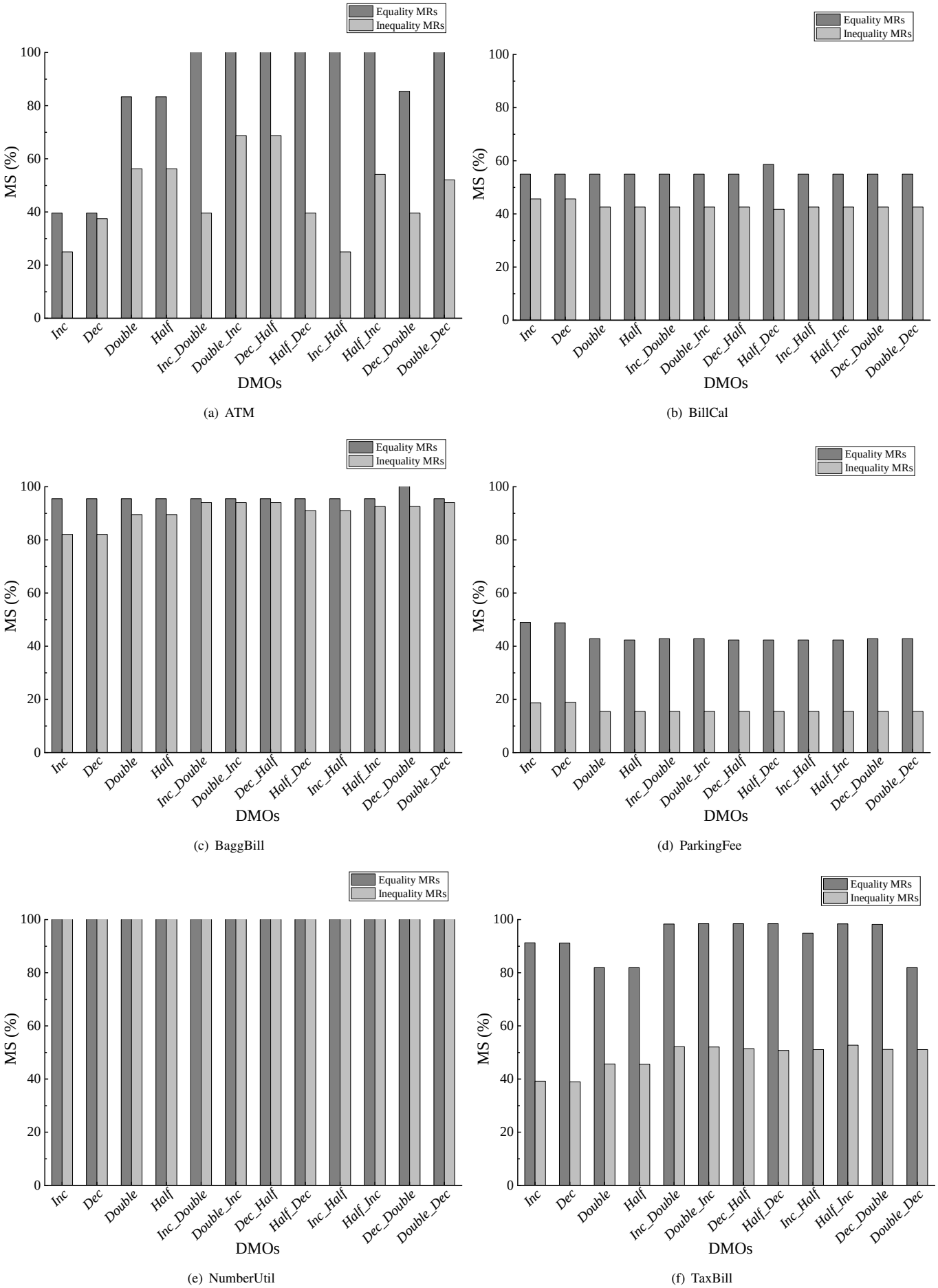
To evaluate the impact of mapping rules on fault detection effectiveness of identified MRs, we conducted an extra experiment where MRs with two types of output relations were identified, and the size of test suites was 100. The results for each program were reported in Figure 8, where the bars in light grey and in dark grey represent mutation scores of MRs with inequality and equality output relations, respectively. From Figure 8, we observed the following:

- Mutation scores of the identified MRs with equality output relations were higher than that with inequality ones across all subject programs except NumberUtil. Specifically, the gaps were significant for ATM, ParkingFee, and TaxBill, while slightly distinct for BillCal and BaggBill. However, for NumberUtil, no distinction was demonstrated since both achieved mutation scores of 100%, that is, four real-life faults of this program could be detected by any of the MRs identified by μMT.
- The gaps of mutation scores of the identified MRs with different output relations varied with subject programs. Specifically, for ATM, the gaps were drastic, ranging from 2.08% (with $Dec(x_i)$) to 75.00% (with $Inc\_Half(x_i)$). For TaxBill, the gaps were always prominent ranging from 30.87% (with $Double\_Dec(x_i)$) to 52.14% (with $Dec(x_i)$). For ParkingFee and BillCal, the gaps stayed rather stable (around 30% and 15%, respectively). For BaggBill, the gaps were relatively small, with a maximum gaps of 13.34% (with $Inc(x_i)$ and $Dec(x_i)$).

The evaluation results showed that the output relations of MRs identified using μMT significantly impacted the fault detection capability. Accordingly, testers are advised to give higher priorities to those MRs with equality output relations whenever possible to achieve higher fault detection effectiveness of MT.

6.5 | Answer to RQ5.1 (Impact of DMOs on fault detection effectiveness of identified MRs)

Besides the mapping rules, DMOs are another crucial element of the μMT methodology. We further evaluate the impact of DMOs on the fault detection effectiveness of identified MRs. Unless otherwise specified, we only took into account MRs with



(a) ATM

(b) BillCal

(c) BaggBill

(d) ParkingFee

(e) NumberUtil

(f) TaxBill

Figure 8 Mutation scores for the MRs identified using different DMOs

equality output relations, although the following analysis was also applicable to those with inequality output relations. From Figure 8, we have observed the following:

- Mutation scores of the identified MRs varied with DMOs for different programs. For ATM, mutation scores varied drastically with DMOs, that is, the smallest with $Inc(x_i)$ and $Dec(x_i)$, followed by $Double(x_i)$, $Half(x_i)$, and $Dec\_Double(x_i)$, and the largest with the other seven composite DMOs. For TaxBill, mutation scores varied prominently with DMOs, that is, the smallest with $Double(x_i)$, $Half(x_i)$, and $Double\_Dec(x_i)$, followed by $Inc(x_i)$, $Dec(x_i)$, and $Inc\_Half(x_i)$, and the largest with the other six composite DMOs. For BillCal, BaggBill, and ParkingFee, mutation scores for all DMOs were quite close, while no difference was demonstrated for NumberUtil.

- Interestingly, MRs identified with $Inc(x_i)$ had always the same mutation scores as $Dec(x_i)$ across all subject programs. Similarly, this was also true for $Double(x_i)$ and $Half(x_i)$. One plausible reason is that the pair of $Inc(x_i)$ and $Dec(x_i)$ resulted in the same MTGs in the context of MT, and thus their fault detection effectiveness must be the same. Accordingly, only one should be selected from the pair of DMOs when they were applicable to reduce the cost of MT. This also applied to the pair of $Double(x_i)$ and $Half(x_i)$.

In summary, DMOs did affect the fault detection effectiveness of the identified MRs in the experiments. The same fault detection effectiveness achieved by MRs with the two pairs of DMOs further suggested that using more DMOs in μMT may not help testers detect more faults from their SUTs.

6.6 | Answer to RQ5.2 (Correlation of fault detection effectiveness of MRs between composite DMOs and individual DMOs)

To investigate the correlation, we compare the fault detection effectiveness of MRs derived using composite DMOs and their component DMOs. To ease the comparison, we look at those MRs with equality output relations (the similar process applies to those MRs with inequality output relations). From Figure 8, we have observed the following: In the vast majority of cases, mutation scores of MRs with composite DMOs were not smaller than those of MRs with component DMOs. For instance, for ATM, the mutation score of MRs identified with $Inc\_Double(x_i)$ (100%) was larger than that of its component DMOs, namely $Inc(x_i)$ (39.58%) and $Double(x_i)$ (83.33%). However, for ParkingFee, the mutation score of MRs identified with $Inc\_Double(x_i)$ (42.84%) was smaller than that with $Inc(x_i)$ (49.01%), while marginally equal to that with $Dec(x_i)$. The above observation further indicates that the mutation score of MRs identified with a composite DMO is less likely smaller than that with its component DMOs. Consequently, when the restricted number of MRs can be used due to the limited test resources, testers are recommended first to try those MRs identified with composite DMOs.

The fault detection effectiveness of MRs with composite DMOs was strongly correlated with their component DMOs. The tester was recommended to give higher priorities to MRs with composite DMOs to achieve a higher fault detection efficiency due to the diverse impact of DMOs on the identified MRs, as observed in Section 6.5.

6.7 | Threats to Validity

Representativeness of subject programs and faults. The subject programs and faults would affect our experimental results' validity. A total of six subject programs were selected from different application domains for evaluation mainly due to their availability, which reduced the effect of this threat to the experimental results. Although our empirical study did not involve large-size programs, our approach itself is generalizable to larger real-life subjects with millions of LOC with the modularization technique, i.e., dividing the program into small modules that can be independently tested. Nevertheless, more complex subject programs would improve the generalization of the obtained results. In our experiments, two kinds of faults were included for evaluation. The real faults were collected from an open-source project, while mutants simulated artificial faults. These mutants were generated using MuJava, a popular mutation tool, with all applicable mutation operators. However, the numbers of real faults and mutants used in our experiments were relatively small.

Correctness of the implemented prototype tool. The prototype tool mainly involved the support of two sets of DMOs and their mapping rules in μMT. Different individuals have checked the implementation, and thus we were confident that their functions are in line with our requirements. In addition, the prototype tool was integrated as a component with MT4WS, which had been extensively tested in our previous studies.

Selection of baseline techniques. Although a variety of techniques have been recently proposed for identifying MRs, as mentioned in Section 1, it is difficult to include all of them as baselines in our experiment. The reason is that they do not share the same starting point with our approach. Thus, setting a fair comparison between them and our approach is difficult. For instance, Kanewala and Bieman[20] proposed an MR detection technique based on machine learning. However, their approach assumes the availability of a long list of simple functions with predefined metamorphic relations for training the classifier, which is fundamentally different from our approach. In our approach, none of the subjects have predefined metamorphic relations, making the assumptions fundamentally distinct. Another MR detection technique is the MR composition technique[25], which focuses on deriving new MRs by combining existing ones. In contrast, μMT primarily focuses on the identification of MRs rather than the composition of MRs. Nonetheless, μMT leverages composite DMOs to identify a broader range of composite MRs, offering a unique capability. Therefore, a direct comparison between μMT and MR composition techniques may not be suitable, as they address different aspects of MR identification. Domain-specific MR identification techniques are designed to address the challenges of MR identification within specific application domains. For example, Troya et al.[65] proposed a specific MR identification technique for ATL model transitions, which focuses on capturing metamorphic relations specific to that domain. In contrast, μMT aims to provide a more versatile approach by effectively identifying MRs without being restricted to specific applications or domains. It offers a general methodology that can be applied across various domains, making it more adaptable and applicable in diverse testing scenarios. Additionally, category-choice-based techniques, as described in studies such as[23,24], utilize the partitioning of input domains and identify MRs by examining the relationships between outputs for distinct complete test frames. It is important to note that these techniques often require domain expert knowledge to accurately classify or divide the output domain into appropriate categories. Search-based techniques, as exemplified by[18], involve the search for suitable parameter values that can represent MRs in a polynomial form. However, it is worth mentioning that these search-based techniques typically assume the availability of a large number of passing test cases. It is important to note that μMT does not assume the presence of such a large number of passing test cases in its methodology. Tabular-expression-based techniques[66] identify MRs from specifications represented in a tabular expression format, focusing on the relationships between header grids and the main grid. In contrast, μMT adopts a data mutation-based approach, simplifying the MR identification process by using DMOs and mapping rules, differentiating it from these techniques with distinct forms of MRs. By highlighting these differences, it becomes evident that μMT offers a unique perspective and methodology for MR identification, setting it apart from other techniques. Accordingly, the comparison of MR identification involved two baseline techniques, namely random selection and mutational MT. The experimental results would be further improved if more MR identification techniques were included for evaluation. Besides the random strategy, which is a commonly used baseline technique, mutational MT was also selected since both mutational MT and our approach were based on DM, and the corresponding comparison made sense.

Appropriateness of evaluation metrics: Previous studies have extensively used the two evaluation metrics involved in our experiments. Mutation score is a well-known metric to evaluate the fault detection capability of testing techniques. Statement coverage has been commonly used to evaluate the adequacy of test suites generated by testing techniques. Thus, the threat of evaluation metrics to our empirical study was minimized.

7 | RELATED WORK

In recent years, MT has been adopted as a popular technique for addressing the oracle problem, receiving increasing attention from both academia and industry[54]. MR is responsible for validating test results and generating new (follow-up) test cases as a core component of MT. Many efforts have been made to identify MRs from different perspectives. We next discuss representative MR identification techniques and compare them with our proposed approach.

Machine learning-based MR identification techniques: This category of techniques leverages the machine learning models to identify MRs from the source code. A predictive model is first trained using a set of MR-indicating features extracted from the source code, and the trained model is then used to predict the corresponding MRs. For instance, Kanewala et al.[20,19] proposed a machine learning-based technique to automatically identify possible MRs using features extracted from the control flow graphs. Ashok et al.[67] proposed using mutation testing to synthesize large amounts of training data and then employing graph kernels and support vector machines to identify MRs. This category of MR identifications assumes the availability of a large amount of source code and normally requires some programs with known MRs for training the predictive model. In contrast, μMT identifies MRs mainly depending on the specification of SUT.

Category-choice-based MR identification techniques: This category of techniques makes use of a category-choice framework to identify MRs. Chen et al.[23] proposed METRIC to identify MRs based on the notion of category and choice, first divides the input domain into a number of disjoint subdomains represented as complete test frames. Then it identifies MRs by examining whether there exist some relations between their outputs for two distinct complete test frames. Sun et al.[24] proposed an enhanced version of METRIC called METRIC$^+$, which significantly improved the efficacy and automation of METRIC by incorporating an output domain-guided mechanism. A tool called MR-GEN$^+$ was developed to automate METRIC$^+$. The empirical study showed that the effectiveness and efficiency of MR identification were improved compared with METRIC. This category of MR identification techniques leverages the widely adopted Category Partition Method (CPM) [68] to partition input domains and then identify MRs with respect to a pair of subdomains. Nevertheless, μMT only works on input parameters without partitioning input domains and leverages DM to direct the identification of MRs.

Search-based MR identification techniques: MRs for numerical programs are normally expressed as polynomials with parameters. Thus, the identification of MRs is transformed as a problem of optimal parameter search. Zhang et al.[18] proposed a search-based MR inference technique. A set of parameters were used to represent a special class of MRs in a polynomial form. Then the class of particle swarm optimization algorithm was developed to find suitable values for these parameters. Search-based MR identification techniques could be very time-consuming since SUT needs to be executed many times to find the genuine MR. On the contrary, the MR identification using μMT could be faster provided that the tester has expertise on numeric programs under test since μMT identifies MRs based on DMOs and mapping rules rather than the search algorithms.

Data mutation-based MR identification techniques: This category of techniques identifies MRs based on DMO. Zhu et al.[22] proposed a mutational MT to integrate DM and MT, and a testing framework called JFuzz was presented. Similar to JFuzz, μMT presented in this work also leveraged DMOs to derive MRs and meanwhile delivered significant enhancement in that: i) Template-style mapping rules were used to derive the output relations of MRs; ii) More individual DMOs and composite DMOs were presented to identify more diverse MRs and composite MRs; iii) A prototype tool was implemented to improve the efficiency of the MR identification further. Furthermore, extensive empirical studies were conducted to evaluate the MR identification effectiveness and fault detection effectiveness of MRs identified using μMT. The experimental results confirmed that μMT was able to identify more MRs with greater fault detection capability than mutational MT.

Tabular-expression-based MR identification technique: Li et al.[66] proposed a technique for identifying MRs from specifications in a tabular expression format, which provides a structured formal representation of specifications. MRs are identified by leveraging the characteristics of tabular expressions, especially the relationships between the header grids and the main grid. This method is only applicable to specifications in tabular expression format. Both this method and μMT provide a kind of domain-independent approach to MR identification, while different principles (tabular expression for the former, while data mutation for the latter) are used to identify MRs and simplify the MR identification process.

MR composition techniques: This category of techniques derive new MRs by combining existing ones. As a pioneering work, Liu et al.[25] proposed the concept of MR composition and provided a way to derive composite MRs. When the follow-up test cases of an MR can be used as the source test case of another MR, the latter MR is composable to the former one. The experimental results have shown that the composed MRs are highly efficient in fault detection. Qiu et al.[27] conducted a theoretical and empirical analysis of the effectiveness of composite MRs, and proposed guidelines for deriving a composite MR with greater fault detection capability. Xiang et al.[69] proposed a genetic algorithm-based MR composition approach, which regards a composite MR as a composition sequence of individual MRs, and searches for a suitable composition sequence using a genetic algorithm such that the execution outputs of test cases satisfy the composite MR. The experimental results showed that their approach could automatically infer high-quality composite MRs. MR composition delivers a simple and efficient to derive new MRs from existing ones. The starting point of such techniques is the availability of some MRs, which could be identified using our proposed approach. Furthermore, μMT leveraged composite DMOs to identify more composite MRs.

Domain-specific MR identification techniques: Besides the above general MR identification techniques, various MR identification techniques have been proposed for specific application domains. Troya et al.[65] proposed an MR identification technique for ATL model transitions. Their technique took as input the execution of a model transformation and automatically inferred a set of MRs. Blasi et al.[70] proposed an equality MR identification technique for natural language documentation and implemented a support tool called MeMo, consisting of two parts, i.e., MR finder and translator. Ayerdi et al.[71] proposed an MR identification technique for cyber-physical systems and implemented a supporting tool called GAssertMRs which can automatically generate MRs through an evolutionary algorithm. Segura et al.[50] proposed an output-pattern-based MR identification technique for RESTful Web APIs. Totally, six kind of patterns defined as the relations between API responses were proposed to capture the typical MRs for Web APIs. Zhu et al.[42] proposed datamorphic testing for AI applications, which consists of seed makers for test

case generation, datamorphisms for the derivation of new test cases from existing one (i.e., DMOs), and metamorphisms for the assertion of test cases (MR is a kind of metamorphisms). Morphy [72] was developed to support datamorphic testing, which classifies software artifacts involved in software testing into test entities and test morphisms (i.e., mappings between test entities). Under the framework of datamorphic testing, a set of testing strategies was further proposed for testing machine learning applications [73,74]. Different from the above techniques, μMT aims at effectively identifying MRs without restrictions to particular applications or domains, although in this study, it was mainly evaluated with various numeric programs.

8 | CONCLUSION

Identifying MRs is an important yet difficult task for MT. In this study, we have proposed μMT, a data mutation directed approach to identifying MRs. In particular, μMT guides the tester to effectively and efficiently identify MRs by providing a set of data mutation operators and template-style mapping rules, alleviating the difficulties faced by MR identification and improving its effectiveness. We have also developed a tool to support μMT. An empirical study with six subject programs was conducted to evaluate the MR identification effectiveness of μMT and the performance of MRs identified by μMT. The experimental results have shown that μMT is able to identify MRs for various numeric programs effectively, and the identified MRs have a high fault detection capability and statement coverage. In the meanwhile, the observations made from the empirical study also suggested a few further guidelines on the selection of mapping rules and DMOs for practicing μMT to achieve better efficiency, particularly when limited testing resources are available.

For future work, we plan to design DMOs that are applicable to non-numeric inputs. The DMOs and subject programs reported in this study are mainly focused on numeric inputs, which does not necessarily mean that μMT are only applicable to numeric programs. As an attempt, we are interested in supporting the inputs of strings since they are widely used in practice. A feasible solution is to design string-related DMO operators, such as substring, comparison, concatenation, and so on, and explore their possible output relations. In addition, we would explore new MR identification directions, for instance, the possibility of synthesizing machine learning techniques to recommend the output relations of MRs in the context of μMT.

ACKNOWLEDGMENTS

This research is supported in part by the National Natural Science Foundation of China (Grant Nos. 62272037, 61872039, and 61370061), the Fundamental Research Funds for the Central Universities (Grant No. FRF-GF-19-B19), the Beijing Natural Science Foundation (Grant No. 4162040), the Aeronautical Science Foundation of China (Grant No. 2016ZD74004), and City University of Hong Kong (Grant No. 9678180).

Author contributions

Chang-ai Sun initiated the project, proposed the main idea, and wrote parts of the manuscript. Hui Jin wrote part of the manuscript, conducted most of the experiment, and developed the prototype. Siyi Wu wrote part of the manuscript and conducted supplementary experiments. An Fu developed the prototype in part and configured the experiments. Zuoyi Wang was involved part of the experiment and development of the prototype. Wing Kwong Chan was involved in the experimental design and research methodology. All authors reviewed the manuscript.

Financial disclosure

Chang-ai Sun is partially funded by the National Natural Science Foundation of China under Grant (Nos. 62272037, 61872039, and 61370061), Fundamental Research Funds for the Central Universities (Grant No. FRFGF-19-B19), Beijing Natural Science Foundation (Grant No. 4162040), and Aeronautical Science Foundation of China (Grant No. 2016ZD74004). Hui Jin is partially funded by the National Natural Science Foundation of China (Grant No. 61872039). Siyi Wu is partially funded by the National Natural Science Foundation of China (Grant No. 62272037). An Fu is partially funded by the National Natural Science Foundation of China under Grant Nos. 61872039 and 61370061. Zuoyi Wang is partially funded by the National Natural Science Foundation of China (Grant No. 61370061). Wing Kwong Chan is partially funded by the City University of Hong Kong (Grant No. 9678180).

Conflict of interest

We declare that we do not have any commercial or associative interest that represents a conflict of interest in connection with the work submitted.

References

1. Sun CA, Liu Y, Wang Z, Chan W. μMT: A Data Mutation Directed Metamorphic Relation Acquisition Methodology. In: Proceedings of the 1st IEEE/ACM International Workshop on Metamorphic Testing (MET'16), in conjunction with the 38th International Conference on Software Engineering (ICSE'16). IEEE; 2016: 12–18.

2. Myers GJ, Sandler C, Badgett T. *The Art of Software Testing*. New Jersey: John Wiley & Sons . 2004.

3. Hierons RM. Oracles for Distributed Testing. *IEEE Transactions on Software Engineering* 2012; 38(3): 629-641.

4. Harman M, Mcminn P, Shahbaz M, Yoo S. A Comprehensive Survey of Trends in Oracles for Software Testing. *University of Sheffield, Department of Computer Science, Tech. Rep. CS-13-01* 2013.

5. Barr ET, Harman M, McMinn P, M S, Yoo S. The Oracle Problem in Software Testing: A Survey. *IEEE Transactions on Software Engineering* 2015; 41(5): 507-525.

6. Brilliant SS, Knight JC, Ammann PE. On the Performance of Software Testing Using Multiple Versions. In: Proceedings of the 20th International Symposium on Fault-Tolerant Computing (FTCS'90). IEEE; 1990: 408-415.

7. Sim KY, Low CS, Kuo FC. Eliminating Human Visual Judgment from Testing of Financial Charting Software. *Journal of Software* 2014; 9(2): 298–312.

8. Chen TY, Cheung SC, Yiu SM. Metamorphic Testing: A New Approach for Generating Next Test Cases. tech. rep., Department of Computer Science , Hong Kong University of Science and Technology, Hong Kong, Tech. Rep. HKUST-CS98-01; Hong Kong University of Science and Technology: 1998.

9. Chen TY, Tse TH, Zhou ZQ. Fault-based testing without the need of oracles. *Information & Software Technology* 2003; 45(1): 1–9.

10. Sun CA, Fu A, Wang Z, Wen Q, Wu P, Chen TY. Iterative Metamorphic Testing for Web services: Technique and Case Studies. *International Journal of Web and Grid Services* 2020; 16(4): 364–392.

11. Batra G, Sengupta J. An Efficient Metamorphic Testing Technique Using Genetic Algorithm. In: Proceedings of the International Conference on Information Intelligence, Systems, Technology and Management. Springer; 2011: 180–188.

12. Dong G, Guo T, Zhang P. 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). IEEE; 2013: 193–197.

13. Barus AC, Chen TY, Kuo FC, Liu H, Merkel R, Rothermel G. A Cost-Effective Random Testing Method for Programs with Non-Numeric Inputs. *IEEE Transactions on Computers* 2016; 65(12): 3509–3523.

14. Barus AC, Chen TY, Kuo FC, Liu H, Schmidt HW. The Impact of Source Test Case Selection on the Effectiveness of Metamorphic Testing. In: Proceedings of the 1st International Workshop on Metamorphic Testing (MET'16), Co-located with the 38th International Conference on Software Engineering (ICSE'16). IEEE; 2016: 5–11.

15. Alatawi E, Miller T, Søndergaard H. Generating Source Inputs for Metamorphic Testing using Dynamic Symbolic Execution. In: Proceedings of the 1st International Workshop on Metamorphic Testing (MET'16), Co-located with the 38th International Conference on Software Engineering (ICSE'16). IEEE; 2016: 19–25.

16. Sun CA, Liu B, Fu A, Liu Y, Liu H. Path-Directed Source Test Case Generation and Prioritization in Metamorphic Testing. *Journal of Systems and Software* 2022; 183: 111091.

17. Wu P. Iterative Metamorphic Testing. In: Proceedings of the 29th Annual International Computer Software and Applications Conference (COMPSAC'05). IEEE; 2005: 19–24.

18. Zhang J, Chen J, Hao D, et al. Search-based Inference of Polynomial Metamorphic Relations. In: Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering (ASE'14). ACM Press; 2014: 701–712.

19. Kanewala U, Bieman JM, Ben-Hur A. Predicting Metamorphic Relations for Testing Scientific Software: A Machine Learning Approach using Graph Kernels. *Software Testing, Verification and Reliability* 2016; 26(3): 245–269.

20. Kanewala U, Bieman JM. Using Machine Learning Techniques to Detect Metamorphic Relations for Programs without Test Oracles. In: Proceedings of the 24th IEEE International Symposium on Software Reliability Engineering (ISSRE'13). IEEE; 2013: 1–10.

21. Zhou ZQ, Sun L, Chen TY, Towey D. Metamorphic Relations for Enhancing System Understanding and Use. *IEEE Transactions on Software Engineering* 2018; 46(10): 1120–1154.

22. Zhu H. JFuzz: A Tool for Automated Java Unit Testing Based on Data Mutation and Metamorphic Testing Methods. In: Proceedings of the 2nd International Conference on Trustworthy Systems and Their Applications. IEEE; 2015: 8–15.

23. Chen TY, Poon PL, Xie X. METRIC: METamorphic Relation Identification based on the Category-Choice Framework. *Journal of Systems and Software* 2016; 116: 177–190.

24. Sun CA, Fu A, Poon PL, Xie X, Liu H, Chen TY. METRIC$^+$: A Metamorphic Relation Identification Technique Based on Input plus Output Domains. *IEEE Transactions on Software Engineering* 2021; 47(9): 1764-1785.

25. Liu H, Liu X, Chen TY. A New Method for Constructing Metamorphic Relations. In: Proceedings of the 12th International Conference on Quality Software (QSIC'12). IEEE; 2012: 59–68.

26. Dong GW, Xu BW, Chen L, Nie CH, Wang LL. Case Studies on Testing with Compositional Metamorphic Relations. *Journal of Southeast University (English Edition)* 2008; 24(4): 437–443.

27. Qiu K, Zheng Z, Chen TY, Poon PL. Theoretical and Empirical Analyses of the Effectiveness of Metamorphic Relation Composition. *IEEE Transactions on Software Engineering* 2020; 48(3): 1001–1017.

28. Chen TY, Kuo FC, Ma W, et al. Metamorphic Testing for Cybersecurity. *Computer* 2016; 49(6): 48–55.

29. Tian Y, Pei K, Jana S, Ray B. DeepTest: Automated Testing of Deep-Neural-Network-Driven Autonomous Cars. In: Proceedings of the 40th IEEE/ACM International Conference on Software Engineering (ICSE'18). IEEE; 2018: 303–314.

30. Zhang M, Zhang Y, Zhang L, Cong L, Khurshid S. DeepRoad: GAN-based Metamorphic Testing and Input Validation Framework for Autonomous Driving Systems. In: Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering (ASE'18). ACM Press; 2018: 132–142.

31. Zhou ZQ, Sun L. Metamorphic Testing of Driverless Cars. *Communications of the ACM* 2019; 62(3): 61–67.

32. Chan WK, Chen TY, Lu H, Tse TH, Yau SS. Integration Testing of Context-Sensitive Middleware-based Applications: A Metamorphic Approach. *International Journal of Software Engineering & Knowledge Engineering* 2008; 16(5): 677–703.

33. Kuo FC, Chen TY, Tam WK. Testing Embedded Software by Metamorphic Testing: A Wireless Metering System Case Study. In: Proceedings of the 36th IEEE Conference on Local Computer Networks. IEEE; 2011: 291–294.

34. Jiang M, Chen TY, Kuo FC, Ding Z. Testing Central Processing Unit Scheduling Algorithms using Metamorphic Testing. In: Proceedings of the 4th International Conference on Software Engineering & Service Science. IEEE; 2013: 530–536.

35. Chan WK, Cheung SC, Leung KR. A Metamorphic Testing Approach for Online Testing of Service-Oriented Software Applications. *International Journal of Web Services Research* 2007; 4(2): 61–81.

36. Sun CA, Wang G, Mu B, Liu H, Wang ZS, Chen TY. Metamorphic Testing for Web Services: Framework and a Case Study. In: Proceedings of the 9th IEEE International Conference on Web Services (ICWS'11). IEEE; 2011: 283–290.

37. Mai PX, Pastore F, Goknil A, Briand LC. Metamorphic Security Testing for Web Systems. In: Proceedings of the 13th IEEE International Conference on Software Testing, Validation and Verification (ICST'20). IEEE; 2020: 186–197.

38. Mayer J, Guderlei R. On Random Testing of Image Processing Applications. In: Proceedings of the 6th International Conference on Quality Software (QSIC'06). IEEE; 2006: 85–92.

39. Xie X, Ho J, Murphy C, Kaiser G, Xu B, Chen TY. Application of Metamorphic Testing to Supervised Classifiers. In: Proceedings of the Ninth International Conference on Quality Software. IEEE; 2009: 135–144.

40. Murphy C, Kaiser GE, Hu L, Wu L. Properties of Machine Learning Applications for Use in Metamorphic Testing. In: Proceedings of the 20th International Conference on Software Engineering and Knowledge Engineering (SEKE'08). Knowledge Systems Institute; 2008: 867–872.

41. Xie X, Ho JK, Murphy C, Kaiser G, Xu B, Chen TY. Testing and Validating Machine Learning Classifiers by Metamorphic Testing. *Journal of Systems & Software* 2011; 84(4): 544–558.

42. Zhu H, Liu D, Bayley I, Harrison R, Cuzzolin F. Datamorphic Testing: A Method for Testing Intelligent Applications. In: Proceedings of the IEEE International Conference On Artificial Intelligence Testing (AITest'19). IEEE; 2019: 149–156.

43. Le V, Afshari M, Su Z. Compiler Validation via Equivalence Modulo Inputs. In: Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'14). ACM Press; 2014: 216–226.

44. Tao Q, Wu W, Zhao C, Shen W. An Automatic Testing Approach for Compiler based on Metamorphic Testing Technique. In: Proceedings of the 17th Asia Pacific Software Engineering Conference (APSEC'10). IEEE; 2010: 270–279.

45. Donaldson AF, Lascu A. Metamorphic Testing for (Graphics) Compilers. In: Proceedings of the 1st International Workshop on Metamorphic Testing (MET'16). ACM Press; 2016: 44–47.

46. Donaldson AF, Evrard H, Lascu A, Thomson P. Automated Testing of Graphics Shader Compilers. *Proceedings of the ACM on Programming Languages* 2017; 1(OOPSLA, Article: 93): 1–29.

47. Jin L. Automated Functional Testing of Search Engine. In: Proceedings of the International Conference on Software Engineering Workshop on Automation of Software Test (AST'09). IEEE; 2009: 97–100.

48. Zhou ZQ, Xiang S, Chen TY. Metamorphic Testing for Software Quality Assessment: A Study of Search Engines. *IEEE Transactions on Software Engineering* 2016; 42(3): 264–284.

49. Brown J, Zhi QZ, Chow YW. Metamorphic Testing of Navigation Software: A Pilot Study with Google Maps. In: Proceedings of the 51st Annual Hawaii International Conference on System Sciences (HICSS'18). ScholarSpace Electronic Library (AISeL); 2018: 1–10.

50. Segura S, Parejo JA, Troya J, Ruiz-Cortés A. Metamorphic Testing of RESTful Web APIs. In: Proceedings of the 40th IEEE/ACM International Conference on Software Engineering (ICSE'18). IEEE; 2018: 882–882.

51. Murphy C, Raunak MS, King A, et al. On effective testing of health care simulation software. In: Proceedings of the 3rd Workshop on Software Engineering in Health Care (SEHC'11), Co-located with the 33th International Conference on Software Engineering (ICSE'11). IEEE; 2011: 40–47.

52. Chen TY, Huang DH, Tse T, Zhou ZQ. Case studies on the selection of useful relations in metamorphic testing. In: Proceedings of the 4th lberoAmerican Symposium on Software Engineering and Knowledge Engineering (JIISIC'04). ; 2004: 569–583.

53. Asrafi M, Liu H, Kuo FC. 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). IEEE; 2011: 147–156.

54. Chen TY, Kuo FC, Liu H, et al. Metamorphic testing: A review of challenges and opportunities. *ACM Computing Surveys* 2018; 51(1): 4.

55. Shan L, Zhu H. Generating Structurally Complex Test Cases By Data Mutation: A Case Study Of Testing An Automated Modelling Tool. *The Computer Journal* 2009; 52(5): 571–588.

56. Liu H, Kuo FC, Towey D, Chen. TY. How Effectively Does Metamorphic Testing Alleviate the Oracle Problem?. *IEEE Transactions on Software Engineering* 2014; 40(1): 4-22.

57. Sun CA, Dai H, Liu H, Chen TY. Feedback-Directed Metamorphic Testing. *ACM Transactions on Software Engineering Methodology* 2023; 23(1): Article 20.

58. Sun CA, Wang G, Wen Q, Towey D, Chen TY. MT4WS: An Automated Metamorphic Testing System for Web Services. *International Journal of High Performance Computing and Networking* 2016; 9(1-2): 104–115.

59. Sun CA, Wang G, Mu B, Liu H, Wang Z, Chen TY. A Metamorphic Relation-based Approach to Testing Web Services without Oracles. *International Journal of Web Services Research* 2012; 9(1): 51–73.

60. Just R, Jalali D, Ernst MD. Defects4J: A Database of Existing Faults to Enable Controlled Testing Studies for Java Programs. In: Proceedings of the 2014 International Symposium on Software Testing and Analysis (ISSTA'14). ACM Press; 2014: 437–440.

61. Ma YS, Offutt J, Kwon YR. MuJava: An Automated Class Mutation System. *Software Testing, Verification and Reliability* 2005; 15(2): 97–133.

62. Cohen J. *Statistical power analysis for the behavioral sciences (2nd ed.* Statistical power analysis for the behavioral sciences . 1988.

63. Arcuri A, Briand L. 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'20). ACM; 2011: 1-10.

64. Chen TY, Kuo FC, Liu H, Wong WE. Code Coverage of Adaptive Random Testing. *IEEE Transactions on Reliability* 2013; 62: 226-237.

65. Troya J, Segura S, Ruiz-Cortés A. Automated Inference of Likely Metamorphic Relations for Model Transformations. *Journal of Systems and Software* 2018; 136: 188–208.

66. Li J, Liu L, Zhang P. Tabular-expression-based method for constructing metamorphic relations. *Journal of Software: Practice and Experience* 2020; 50(8): 1345–1380.

67. Ashok NA, Meinke K, Eldh S. Leveraging Mutants for Automatic Prediction of Metamorphic Relations using Machine Learning. In: Proceedings of the 3rd ACM SIGSOFT International Workshop on Machine Learning Techniques for Software Quality Evaluation (MaLTeSQuE'19). ACM Press; 2019: 1–6.

68. Ostrand TJ, Balcer MJ. The category-partition method for specifying and generating fuctional tests. *Communications of the ACM* 1988; 31(6): 676–686.

69. Xiang Z, Wu H, Yu F. A Genetic Algorithm-Based Approach for Composite Metamorphic Relations Construction. *Information* 2019; 10(12): 392–407.

70. Blasi A, Gorla A, Ernst MD, Pezzè M, Carzaniga A. MeMo: Automatically Identifying Metamorphic Relations in Javadoc Comments for Test Automation. *Journal of Systems and Software* 2021; 181: 111041.

71. Ayerdi J, Terragni V, Arrieta A, Tonella P, Sagardui G, Arratibel M. Generating Metamorphic Relations for Cyber-Physical Systems with Genetic Programming: An Industrial Case Study. In: Proceedings of the 29th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE'21). ACM Press; 2021: 1264–1274.

72. Zhu H, Bayley I, Liu D, Zheng X. Automation of Datamorphic Testing. In: Proceedings of 2020 IEEE International Conference On Artificial Intelligence Testing (AITest'20). IEEE; 2020: 64–72.

73. Zhu H, Bayley I. Exploratory Datamorphic Testing of Classification Applications. In: Proceedings of 1st IEEE/ACM International Conference on Automation of Software Test (AST'20). ACM Press; 2020: 51–60.

74. Zhu H, Bayley I. Discovering boundary values of feature-based machine learning classifiers through exploratory datamorphic testing. *Journal of Systems and Software* 2022; 187: 111231.