A Practical and Efficient Metamorphic Relation Acquisition Approach via CPM

Chang-ai Sun*, An Fu*, Huai Liu†, Tsong Yueh Chen‡, Xiaoyuan Xie§, Pak-Lok Poon¶
*School of Computer & Communication Engineering, University of Science and Technology Beijing, Beijing 100083, China
†Australia-India Research Centre for Automation Software Engineering, RMIT University, Melbourne, Australia
‡Department of Computer Science and Software Engineering, Swinburne University of Technology, Melbourne, Australia
§State Key Lab of Software Engineering, Computer School, Wuhan University, China
¶School of Business IT and Logistics, RMIT University, Melbourne, Australia

***Abstract*—Metamorphic testing is a software testing technique which can effectively alleviate the oracle problem. The main idea of metamorphic testing is to test a software by checking whether a metamorphic relation holds among several executions. How to acquire metamorphic relations from the software under test is a very essential task in metamorphic testing. METRIC (Metamorphic Relation Identification Based on Category-choice Framework) is a systematic technique to help software testers acquire metamorphic relations. This technique takes the input of software under test into consideration and uses candidate pairs formed by complete test frames to identify metamorphic relations. This paper believes that the output is an important aspect of a software and should also be considered during a test procedure. In this paper, we proposed an improved approach named METRIC* to improve METRIC in terms of effectiveness and efficiency through introducing the output of software. A tool implementing METRIC* is developed. An empirical study involving four real-life specifications is conducted to evaluate the effectiveness and efficiency of METRIC*. The results have shown that METRIC* improves the correctness and speed of metamorphic relations identification and is a practical and efficient metamorphic relations acquisition approach. The fault detection effectiveness of metamorphic relations identified is investigated by comparing with another approach called μMT. The results indicate that metamorphic relations identified via METRIC* have good fault detection capability.**

***Index Terms*—Metamorphic Testing, Metamorphic Relation, Category-Choice Framework, Fault Detection Effectiveness.**

I. Introduction

Software testing is an important approach to guarantee the quality of software systems. When quality of software systems refers to the correctness of functions, we test whether the software can do the right thing in a right way. In general, a software is considered to pass a test if it behaves as expected during the test procedure. Otherwise, it is considered to fail a test. The mechanism for determining a test has passed or failed is called an oracle [1]. A common approach is to compare the actual output with expected output. For many software testing techniques, it is assumed that the oracle of a software under test exists. But, under certain circumstances, oracle does not exist or is not feasible, which is so called the oracle problem. When oracle problem exists, testing techniques requiring oracles are hard to be applied.

To address the oracle problem, various techniques have been proposed such as model checking [2], assertion [3], and n-version programming [4]. Metamorphic testing [5] is a software testing technique that aims at situations when oracle problem exists. This technique uses metamorphic relations which can be derived from characteristics of a software to detect faults. For a given software, we can derive some essential properties according to its specification that reflect the inner relationship among several tests. These properties are commonly known as metamorphic relations and is expected to be hold between the specification and its corresponding implementation. Otherwise, the implementation is said to be faulty. Throughout the testing procedure, multiple executions are involved to check whether metamorphic relations are satisfied instead of validating the expected output. Thus, oracle problem is alleviated.

Given the advantage of metamorphic testing, more and more researchers and software testers are beginning to focus on this promising testing technique. It is reported that with a small number of diverse metamorphic relations, metamorphic testing could effectively help alleviate the oracle problem [6]. Paper [7] reviewed recent research of metamorphic testing and addressed open challenges. Paper [8] summarized research results and presented challenges and future improvements of metamorphic testing. Metamorphic testing has now been successfully used in various fields including compiler validation [9], bioinformatics [10] [11], embedded systems [12] [13], cybersecurity [14], and web services [15] [16].

While metamorphic testing is receiving more and more attentions, some limitations still exists such as acquisition of good metamorphic relations, effective test case generation, and research on foundation theory of metamorphic testing, making its development and application hampered. Among these limitations, metamorphic relation acquisition is a challenging task. Most metamorphic relation identification is done manually or arbitrarily [8] and even experts may find it difficult to acquire metamorphic relations [17]. Metamorphic relation plays an important role in metamorphic testing, without which it is not able to apply metamorphic testing. If metamorphic relations cannot be identified or generated effectively and efficiently, the application of metamorphic testing would be limited.

To address this problem, a number of techniques are proposed such as machine learning based metamorphic relation detection [18], metamorphic relation composition [18], search-based metamorphic relation inference [19], and data mutation directed metamorphic relation acquisition [20]. Chen et al. developed a systematic and effective metamorphic relation identification methodology named METRIC [21]. METRIC uses complete test frames generated by category choice framework [22] according to the input of a software to help testers obtain metamorphic relations. This methodology enables testers to form candidate pairs using complete test frames and to decide whether these candidate pairs can be used to derive metamorphic relations. METRIC takes the input of software into consideration and helps testers identify metamorphic relations from specifications in a systematic way. However, limitations such as lack of candidate pair selection strategy and poor guidelines restrict the effectiveness and efficiency.

This paper aims at improving METRIC in terms of effectiveness and efficiency by introducing software output. Our goals are to provide an explicit candidate pair selection mechanism and to make metamorphic relation identification much easier and more systematically. We believe that not only the input is an important perspective, but also the output is an important perspective. It is not comprehensive to consider the input domain only. Thus, software output domain is partitioned and introduced into METRIC. A candidate pair selection mechanism is provided to reduce the number of candidate pairs that need to be decided. To study the difference between METRIC and METRIC*, we conducted an empirical study and evaluated these two approaches in terms of effectiveness and efficiency. Furthermore, we would like to know whether metamorphic relations identified via METRIC* can reveal faults. We studied the fault detection capability of metamorphic relations identified via METRIC* and compared these metamorphic relations with those identified via μMT.

The main contributions of this paper are:

1) An improved metamorphic relation acquisition approach named METRIC* is proposed. An explicit selection mechanism of candidate pair is provided to reduce the search space of candidate pair.
2) A supporting tool named MR-GEN* is developed, which automates parts of METRIC* and helps testers with metamorphic relation identification.
3) An empirical study involving four real-life specifications is conducted to study the effectiveness and efficiency of METRIC*.
4) Fault detection effectiveness of metamorphic relations identified via METRIC* is investigated and compared with that of metamorphic relations identified via μMT.

The rest of this paper is organized as follows: Section 2 introduces the concepts of metamorphic testing, category-choice framework and METRIC. Section 3 explains the motivations of introducing output and methodology of METRIC*. Section 4 describes the details of a supporting tool named MR-GEN*. Section 5 describes the setting of empirical study. Section 6 reports the results and discussions of empirical study. Section 7 discusses limitations and potential threads to the study. Section 8 discussed some related work. Section 9 summarizes this paper.

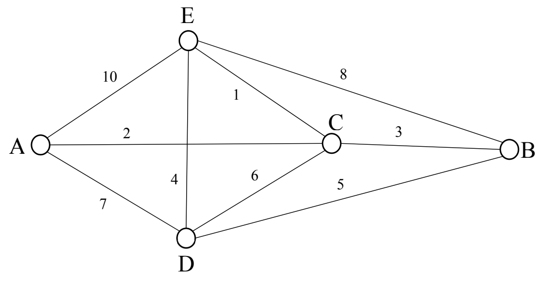


Fig. 1: An input graph for Example 1

II. Background Knowledge

A. Metamorphic Testing

Metamorphic testing was proposed by Chen et al. in 1998, which is still applicable in absence of oracle when testing a software. The basic steps of metamorphic testing are as follows:

1) Identify necessary properties of the software under test and derive relations between multiple inputs and their corresponding outputs, commonly known as metamorphic relations.
2) Generate test cases using existing test case generation techniques. These test cases are referred to as source test cases.
3) Generate follow-up test cases from source test cases according to metamorphic relations derived before.
4) Apply source test cases and their corresponding follow-up test cases to the software under test and check whether a metamorphic relation holds among the outputs of relation's relevant source test cases and follow-up test cases.

The following example illustrates the detailed process of metamorphic testing.

Example 1: (Shortest Path in an Undirected Graph). Consider a program SP that can search for the shortest path between two specific nodes in a given undirected graph like Figure 1. We use $SP(G, A, B)$ to represent the shortest path found from point A to point B in graph G. When graph G is complicated, it is not easy to verify the correctness of program output. But there are some properties within program SP that we can make use of. It is very easy to know that if we swap the start node and the end node, the length of shortest path between two nodes will not change. This property can be used to derive the changes of output when input is changed in a specific way, thus it can be considered as a metamorphic relation. If we construct test cases that satisfy the above differences in terms of input, then the output of these test cases should theoretically be the same. If we apply these test cases to the program under test and find out that the outputs of these test cases are not the same, then we have confidence that the program under test is faulty. Consider a source test case

(G, A, B). According to the metamorphic relation, follow-up test case (G, B, A) is constructed. After applying these two test cases to execute the program, we decide whether $|SP(G, A, B)|$ equals to $|SP(G, B, A)|$. If metamorphic relation is not satisfied, then program SP is faulty. Another property within program SP which we can make use of is that if point C is a node in the shortest path between point A and point B, then the length of shortest path from A to B must be the sum of shortest distance from A to C and from C to B. The inner metamorphic relation can be described as follow: $|SP(G, A, B)| = |SP(G, A, C)| + |SP(G, C, B)|$ where C is a node in $SP(G, A, B)$. According to this relationship we can derive two follow-up test cases, (G, A, C) and (G, C, B), from (G, A, B). After the execution of these test cases, the output can be verified by checking whether above metamorphic relation is satisfied.

Above example shows that expected output of test cases were not involved in the whole testing procedure, thus metamorphic testing can alleviate the oracle problem and can effectively test programs when oracle problem occurs. Commonly, a metamorphic relation is consist two parts: a sub-relation on input and a sub-relation on output. As for the first property within the example above, the sub-relation on input, denoted as "r", can be described as follow: the start point and the end point is swapped. The sub-relation on output, denoted as "rf", can be described as follow: the length of shortest path do not change.

B. CHOC'LATE and CHOC'LATE-DIP

CHOCLATE [22] is a test case generation framework. This framework makes use of categories, choices and complete test frames to generate test cases from software specifications. The concepts of category, choice and complete test frame are as follows:

- Category: a category is a major property or characteristic of a functions parameter or an environment condition that affect the execution behavior of the function.
- Choice: the value domain of a category is partitioned into several disjoint sub-domains named choices.
- Complete test frame: a complete test frame is a valid choice combination formed by one choice in each category. Complete test frames are generated according to constrains among choices.

By selecting a single element from each choice contained in a complete test frame, the set of elements will constitute a test case.

Choice relation framework with distinguishing output scenarios (CHOC'LATE-DIP) [23] is a partition testing technique based on CHOC'LATE. By introducing the output scenario, CHOC'LATE-DIP improves CHOC'LATE from the following perspectives:

- Different scenarios of program outputs are identified and refined into categories and choices. These categories and choices related to program outputs are referred to as O-categories and O-choices respectively. Categories and choices which related to program inputs are referred to as I- categories and I-choices.
- Choice relation table including I-choices and O-choices is constructed. This extended choice relation table not only contains the relation between each I-choice pair, but also contains relation between each O-choice pair as well as that between every I-choice and O-choice.
- Complete test frames containing I-choices and O-choices are constructed based on extended choice relation table. These complete test frames which contains information about program output are referred to as IO-based complete test frame (IO-CTF).
- When generating test cases from IO-CTF, the type of expected output can also be determined. This advantage makes it easier for software testers to determine the expected output corresponding to a test case.

As a reminder, a IO-CTF composed of input choices including "a", "b", and a output choice "c" is denoted as "{a, b; c}" in this paper.

C. METRIC

Normally, when identifying a metamorphic relation, we need to consider how the output changes when the input is changed. Simply speaking, we need to change software inputs first and then get the corresponding predictable change in output. So how to fulfill the above objective? One way is to change inputs directly and then predicate changes in output according to specifications, which is simple but undirected. Another way is to make comparison between different inputs, and predict changes in terms of output. But this will also bring problems. How to acquire different inputs is an essential task. A common approach is to generate different test cases using existing test case generation method, but this will bring another problem. A large number of test cases might be generated and a number of comparisons need to be done, which is not feasible. The category-choice framework can generate complete test frames from which test cases can be generated. Each complete test frame represents an input scenario and therefore more abstract than test cases. It is feasible to compare test frames rather than compare test cases directly. Based on the above ideas, Chen et al. proposed metamorphic relation identification based on category choice framework (METRIC). METRIC is a systematic methodology which enables software testers to identify metamorphic relations. This technique makes use of complete test frames generated by category-choice framework according to software specifications and guide software testers in an easy and brief way. The basic steps of METRIC are as follows:

1) Two distinct complete test frames are selected to form a candidate pair.
2) Enable the software tester to determine whether the current candidate pair implies a metamorphic relation. If so, enable the software tester to describe the inner metamorphic relation.
3) Restart from step 1), and repeat until all the complete test frames are compared with any other complete test frames or the expected number of identified metamorphic relations is reached.

Candidate pairs that implies metamorphic relations are said to be usable, and those that do not implies metamorphic relations are said to be unusable.

METRIC enable testers to concentrate on only one pair of complete test frame and determine the relationship between the outputs, which is far easier than the traditional way of considering both input and output aspects.

III. METRIC* METHODOLOGY

A. Motivation

Partition testing has been receiving increasing attention for its simplicity and usability. A common approach is to divide all possible program inputs into disjoint partitions, from which test cases are selected. The idea is that all test cases within a partition shall be homogenious. So theoretically, a single test case will be sufficient to represent a partition. But when the partition is not homogeneous, then the statement above will be faulty. Huai et al. discovered that partition testing could be enhanced by considering the variation of output. Their studies have shown that if output is introduced in complete test frame generated by category-choice framework, the partition can be further subdivided according to different outputs.

Based on the above observation, we can get an intuition that METRIC, built upon category-choice framework, can be enhanced through introducing software output into complete test frames. Simply speaking, when identifying metamorphic relations, the use of IO-CTF has a great advantage than original complete test frames. The reasons are as follows:

1) IO-CTF contains information related to the output compared to the original complete test frame. When determining candidate pairs, software testers can directly see the changes that occur in output rather than predicting those changes through original complete test frames.
2) The introduction of output enables testers to group complete test frames. IO-CTFs are classified into several groups of which the combination of output choices are the same. After IO-CTFs are grouped, it is possible to reduce unusable candidate pairs. As shown in Figure 2, suppose group A has 3 IO-CTFs and their corresponding output is “output1”, group B has 3 IO-CTFs and their corresponding output is “output2”. Select a complete test frame from group A and select a complete test frame from group B to form a candidate pair. When all combinations are exhausted, nine candidate pairs are generated. If METRIC is used to identify metamorphic relations, all nine candidate pairs need to be decided. When output is introduced and IO-CTFs are grouped, we first compare group A with group B and decide whether we can derive output relations through comparing “output1” and “output2”. If not, then it is obvious that these nine candidate pairs are not useable for metamorphic relation identification, thus unusable candidate pairs are abandoned and total number of candidate pairs that need to be decided is reduced. Actually, software output is used to extract the commonality within candidate pairs and makes it possible to identify unusable candidate pairs.
3) More automatic preparation and systematic guidelines are introduced at the same time. The grouping of IO-CTFs and the reduction of candidate pairs can be done in a more automatic way.

B. Methodology

Through introducing extra O-categories and O-choices, the identification of metamorphic relations is more directly. Furthermore, it is possible to group IO-CTFs and reduce unusable candidate pairs.

In general, METRIC* involves the following tasks:

1) Construct the complete test frames (IO-CTF), containing both I-category/I-choice and O-category/O-choice, that is, each IO-CTF is a possible combination of I-choices and O-choices;
2) Classify all IO-CTFs into groups of which the O-choice combinations are the same. These groups are called test frame groups. Within a test frame group, all IO-CTFs have the same combination of O-choices.
3) Identify metamorphic relations through candidate pairs formed by IO-CTFs with the same O-choice combination:
    - Select a test frame group and enable software tester to determine whether the output relation, denoted as Ro, exists when the O-choice combination remains unchanged.
    - If Ro exists and the number of IO-CTFs belonging to current test frame group is more than 1, then select two IO-CTFs within current test frame group that are distinct in the combination of I-choices to form a candidate pair for user’s consideration. Enable the tester to determine the input relation, denoted as Ri, between the I-choice combinations of two IO-CTFs. Finally a metamorphic relation is defined as Ri plus Ro. If current test frame group has only one IO-CTF, then change another group and restart from task 3. Repeat this step until every IO-CTF has combined with any other IO-CTF within current test frame group.
    - If Ro does not exist, then change another group and restart from task 3.
4) Identify metamorphic relations through candidate pairs formed by IO-CTFs with different O-choice combinations:
    - Select two test frame groups and enable the tester to determine whether the output relation, denoted as Ro, exists between O-choice combinations of these two test frame groups.
    - If Ro exists, then select two IO-CTF, each of which is from one of the test frame groups, respectively, as a candidate pair for tester’s consideration. Enable the tester to determine the input relation, denoted as Ri, between I-choice combinations of these two IO-CTFs. Finally a metamorphic relation is defined as Ri plus Ro. Repeat this step until every IO-CTF within a test frame group has combined with every IO-CTF within another test frame group.

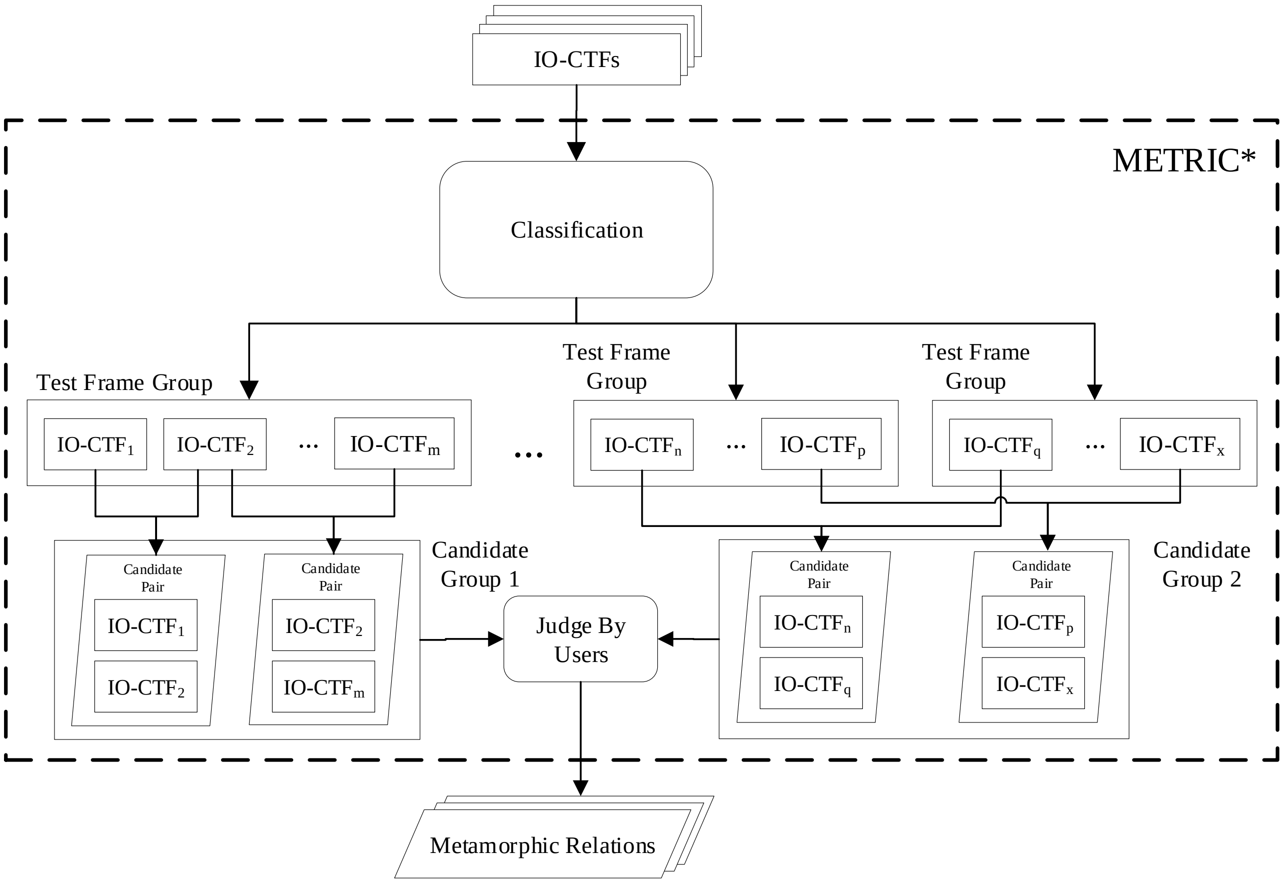


Fig. 2: Overview of METRIC*

- If Ro does not exist, then change another two groups and restart from task 4.

When Ro does not exist in task 3, it indicates that the change of output cannot be determined. Thus the output relation of candidate pairs formed by any two IO-CTFs within current test frame group cannot be determined, which means that these candidate pairs are unusable. Then there is no need to select two IO-CTFs within this test frame group and form candidate pairs. In task 4, the reason is the same. Simply speaking, unusable candidate pairs are identified and abandoned in advance. As for METRIC, a tester has to determine every candidate pair and determine whether they are useful.

An example is proposed to demonstrate how METRIC* works.

Example 2: (Credit Card System). Consider a credit card system, which processes customer's operation request and output whether the request is accepted or rejected. The input of this credit card system includes card number, status, manager's special permission, credit limit and the amount of operation. The basic business process of the system are as follows:

1) Check whether the card number exists. If it exists, go to step 2). Otherwise, return "Rejected";
2) Check the status of credit card. If the credit card is "activated", go to step 4). Otherwise, go to step 3);
3) If the credit card has manager's special permission, then return "Accepted". Otherwise, return "Rejected";
4) If the amount of operation is larger than credit limit, then return "Rejected". Otherwise, return "Accepted".

TABLE I: Input Categories and Choices of Example 2

I-category	I-choice
1. Card number	1a. Exists
	1b. Do not exist
2. Status	2a. Activated
	2b. Not activated
3. Manager's special permission	3a. Yes
	3b. No
4. Amount of operation	4a. $\leq$ credit limit
	4b. $\geq$ credit limit

TABLE II: Output Categories and Choices of Exmaple 2

O-category	O-choice
I. Card number	Ia. Accepted
	Ib. Rejected

According to CHOC'LATE-DIP, we can acquire categories and choices shown in Table 1 and Table 2.

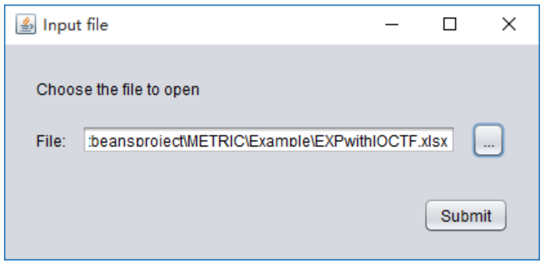
After identifying categories and defining choice, IO-CTFs are generated through combining choices from different categories. All IO-CTFs generated using CHOC'LATE-DIP are

TABLE III: IO-based Complete Test Frames

ID	IO-CTF	ID	IO-CTF
1	{1a,2a,3a,4a;Ia}	2	{1a,2a,3a,4b;Ib}
3	{1a,2a,3b,4a;Ia}	4	{1a,2a,3b,4b;Ib}
5	{1a,2b,3a,4a;Ia}	6	{1a,2b,3a,4b;Ia}
7	{1a,2b,3b,4a;Ib}	8	{1a,2b,3b,4b;Ib}
9	{1b,2a,3a,4a;Ib}	10	{1b,2a,3a,4b;Ib}
11	{1b,2a,3b,4a;Ib}	12	{1b,2a,3b,4b;Ib}
13	{1b,2b,3a,4a;Ib}	14	{1b,2b,3a,4b;Ib}
15	{1b,2b,3b,4a;Ib}	16	{1b,2b,3b,4b;Ib}

TABLE IV: IO-CTF Groups

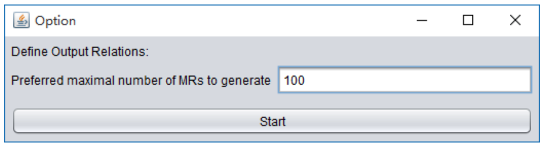
Group Name	IO-CTF	
Ia	{1a,2a,3a,4a;Ia}	{1a,2a,3b,4a;Ia}
	{1a,2b,3a,4a;Ia}	{1a,2b,3a,4b;Ia}
Ib	{1a,2a,3a,4b;Ib}	{1a,2a,3b,4b;Ib}
	{1a,2b,3b,4a;Ib}	{1a,2b,3b,4b;Ib}
	{1b,2a,3a,4a;Ib}	{1b,2a,3a,4b;Ib}
	{1b,2a,3b,4a;Ib}	{1b,2a,3b,4b;Ib}
	{1b,2b,3a,4a;Ib}	{1b,2b,3a,4b;Ib}
	{1b,2b,3b,4a;Ib}	{1b,2b,3b,4b;Ib}


Input file

Choose the file to open

File: :beansproject\METRIC\Example\EXPwithIOCTF.xlsx

Fig. 3: Screen for file selection


Option

Define Output Relations:

Preferred maximal number of MRs to generate 100

Fig. 4: Screen for defining maximum number of MRs

shown below.

Next, IO-CTFs are classified into groups of which the O-choice combinations are the same. Test frame groups and their corresponding IO-CTFs are shown below.

We first identify metamorphic relations within each test frame group. It is obvious that two test frame groups need to be identified. We choose Group Ia first. The corresponding O-choice of Group Ia is “Accepted”. When the O-choice is “Accepted” and holds, it is easy to figure out that output do not change and remains “Accepted”. After that, we select two IO-CTFs from Group Ia and determine input relation. Suppose {1a,2a,3a,4a;Ia} and {1a,2a,3b,4a;Ia} are selected. Comparing the I-choice part, the input relation is determined as follow: The number of credit card exists, the status of credit card is activated and the amount of operation is under credit limit, but special permission is changed. Finally a metamorphic relation is defined as input relation plus output relation. Detail metamorphic relation is described as follow: In the case where the card number exists, the status of credit card is activated and the amount of operation is less than credit limit, when special permission is changed, the output will not change and remains “Accepted”. Repeat above steps until every IO-CTF has combined with any other IO-CTF within Group Ia. When dealing with Group Ib, processes are the same.

We identify metamorphic relations across two different test frame groups next. Two test frame groups are generated and thus it is only able to select these two test frame groups for tester’s consideration. The O-choice of Group Ia is “Accepted” while O-choice of Group Ib is “Rejected”. When the O-choice is changed, the output of program under test is changed. After that, select one IO-CTF from Group Ia select one IO-CTF from Group Ib to form a candidate pair for software tester’s consideration. Suppose {1a,2a,3a,4a;Ia} and {1a,2a,3a,4b;Ib} are selected. A tester can determine the input relation through comparing the I-choice part. The input relation is described as follow: the number of credit card exists, the status of credit card is activated and the credit card has manager’s special permission, but the amount of operation changes from no more than credit limit to higher than credit limit or the opposite. Finally a metamorphic relation is defined as input relation plus output relation. Detail metamorphic relation is described as follow: In the case where number of credit card exists, status of credit card is activated and credit card has manager’s special permission, when the amount of operation changes from no more than credit limit to higher than credit limit or the opposite, the output will change oppositely. Repeat above steps until every IO-CTF within group Ia has combined with every IO-CTF within Group Ib.

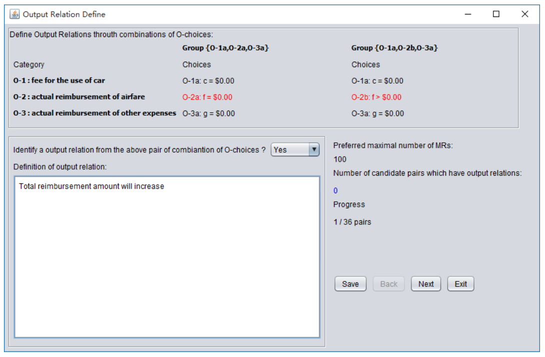
IV. Supporting Tool

Section 4 describes the details of a supporting tool named as MR-GEN*. Based on MR-GEN, a supporting tool called MR-GEN* has been developed. New features including abilities to process output related categories and choices, group IO-CTFs, identify and abandon unusable candidate pairs have been integrated into the supporting tool.

MR-GEN* acquires categories, choices, and IO-CTFs of software under test through a spreadsheet. Data in this spread sheet is stored in a specific form which is easy for software testers to view and modify. MR-GEN* allows the tester to specify the maximum number of metamorphic relations to be identified.

After the spreadsheet is read and parsed, pairs of test frame groups with their corresponding O-choice combinations are shown one by one. Differences between two O-choice combinations are marked in red, helping software testers to determine whether an output relation is implied. If output relation exists, MR-GEN* enables testers to write down the relation and record it. Current pair of test frame group will be marked as usable and relevant candidate pairs will be generated for input relation determination. If output relation does not exists. MR-GEN* enables testers to mark this pair of test frame group as unusable and relevant candidate pairs will not be generated.

When all pairs of test frame groups are determined and candidate pairs are generated, all usable candidate pairs with their corresponding I-choice combinations are shown one by



Define Output Relations throuth combinations of O-choices:

	Group {O-1a,O-2a,O-3a}	Group {O-1a,O-2b,O-3a}
Category	Choices	Choices
O-1 : fee for the use of car	O-1a: c = $0.00	O-1a: c = $0.00
O-2 : actual reimbursement of airfare	O-2a: f = $0.00	O-2b: f > $0.00
O-3 : actual reimbursement of other expenses	O-3a: g = $0.00	O-3a: g = $0.00

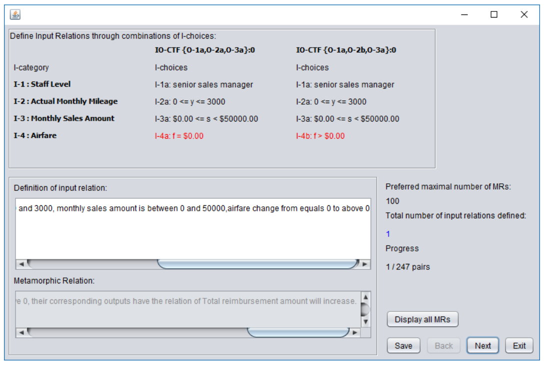
Identify a output relation from the above pair of combiantion of O-choices ? Yes

Definition of output relation:

Total reimbursement amount will increase

Preferred maximal number of MRs:
100
Number of candidate pairs which have output relations:
0
Progress
1 / 36 pairs

Fig. 5: Screen for defining Output Relations



Define Input Relations through combinations of I-choices:

	IO-CTF {O-1a,O-2a,O-3a}:0	IO-CTF {O-1a,O-2b,O-3a}:0
I-category	I-choices	I-choices
I-1 : Staff Level	I-1a: senior sales manager	I-1a: senior sales manager
I-2 : Actual Monthly Mileage	I-2a: 0 <= y <= 3000	I-2a: 0 <= y <= 3000
I-3 : Monthly Sales Amount	I-3a: $0.00 <= s < $50000.00	I-3a: $0.00 <= s < $50000.00
I-4 : Airfare	I-4a: f = $0.00	I-4b: f > $0.00

Definition of input relation:

and 3000, monthly sales amount is between 0 and 50000,airfare change from equals 0 to above 0

Metamorphic Relation:

e 0, their corresponding outputs have the relation of Total reimbursement amount will increase.

Preferred maximal number of MRs:
100
Total number of input relations defined:
1
Progress
1 / 247 pairs

Fig. 6: Screen for defining input relations

one. Also differences between two I-choice combinations are marked in red. Software testers can make use of the difference to determine the input relation that two I-choice combinations implies. MR-GEN* enables testers to write down the input relation. When input relation of a candidate pair is defined, metamorphic relation is constructed by combining input relation and output relation. Testers can click "Display all MRs" to view the already defined metamorphic relations.

V. Experiment

In section 3, the methodology of METRIC* has been illustrated and the principle of improvement has been described. However, the practical performance of METRIC* remains to be studied. Whether METRIC* is as expected is still unknown. Thus, it is essential to evaluate the proposed method in reality. In this section, an empirical study is conducted to study the difference between METRIC and METRIC* in terms of effectiveness and efficiency.

After identifying metamorphic relations, whether these relations are helpful for fault detection remains unknown. If these metamorphic relations can effectively detect faults, then METRIC* is meaningful. Otherwise, METRIC* will not be helpful for fault detection. Also, it is a meaningful research to study the type of faults that are difficult to be detected by metamorphic relations identified by METRIC*, which can help us to discover the weakness of such metamorphic relations.

A. Research questions

The aim of this empirical study is to answer the following research question:

RQ1 **Can METRIC* be practically used for MR identification?**
Even though METRIC* and METRIC are very similar in terms of metamorphic relation identification mechanism, the differences between these two approach are also obvious. METRIC* makes use of software output but METRIC dont. Another change is that METRIC* can group IO-CTFs according to O-choice combinations but METRIC dont. These changes might weaken the ability of METRIC* in terms of identification of metamorphic relations. So whether METRIC* is able to identify metamorphic relations is a meaningful study. Four commercial software specifications were used in the experiment. The goal is to observe whether METRIC* can successfully identify metamorphic relations from these specifications. Participants involved in the experiment were graduate students whose major was software engineering.

RQ2 **Dose the use of METRIC* result in a difference in MR identification compared with METRIC?**
To answer this research question, software testers are conducted to identify metamorphic relations using METRIC and METRIC*. We evaluate the performances of software testers in the process of metamorphic relation identification. By comparing the performances of software testers under different methodologies, we are able to study the difference between METRIC and METRIC* in terms of effectiveness and efficiency.

RQ3 **Can METRIC* reduce the search space of candidate pairs that need to be decided**
Theoretically, METRIC* can identify and abandon unusable candidate pairs in advance, thus reduce the number of candidate pairs that need to be decided by software testers. But theory and practice are two different things. Whether METRIC* can reduce the search space of candidate pairs that need to be decided remains unknown. To study this problem, we identified all candidate groups generated by METRIC* and recorded the number of candidate pairs which was abandoned by METRIC*. We use "Num" to represent the number of candidate pairs that need to be decided when METRIC is applied. We use "Num*" to represent the number of candidate pairs that need to be decided when METRIC* is applied. The goal is to compare "Num *" with "Num" and see whether the former is less than the latter.

RQ4 **How effectively can a metamorphic relation set which is formed by all metamorphic relations identified through METRIC* detect software faults?**
Metamorphic relation is the key part of metamorphic testing. The fault detection capability of metamorphic relations set has a great influence on the effectiveness of metamorphic testing. To answer this research question, the fault detection effectiveness of metamorphic relations sets was studied through mutation analysis. The mutants killed by metamorphic relations sets are counted and mutation scores of metamorphic relation sets are calculated. A series test suits which are of different size are generated and used to study the impact of test suits size on fault detection capability of metamorphic relations

set. Finally the fault detection capability of metamorphic relation sets identified through METRIC* was compared with metamorphic relation sets identified through μMT.

RQ5 **Which faults are difficult to be detected by metamorphic relations identified through METRIC*?**
By answering this research question, shortcomings of metamorphic relations identified through METRIC* in terms of fault detection can be revealed. To study this question, mutants that could not be killed by all metamorphic relations were recorded and analyzed. The reasons why these mutants could not be killed were further investigated.

B. Specifications

As for RQ1 and RQ2, four commercial software specifications were used in the experiment. SPBC, which is the specification of a cellphone billing system related to a telecommunications company; SBBS, which is the specification of a baggage billing service related to an airline company; SCAR, which is the specification of a company car and expense claim system; and SMOS, which is the specification of a meal ordering system related to a company providing catering services for different airlines. Categories and choices of software input and output were identified by the tutorial previously. Complete test frames used by MR-GEN were generated according to input categories and choices. As for MR-GEN*, IO-CTFs, which involved software output, were generated according to input and output categories and choices. Complete test frames (IO-CTFs, respectively) of a specification with its relevant categories and choices were recorded in an excel file, which served as the input of MR-GEN (MR-GEN*, respectively). As a reminder, complete test frames were randomly selected by MR-GEN. The selected complete test frame was then provided to participants to identify metamorphic relations. As for METRIC*, the situation is similar. During the stage of output relation definition, candidate groups are randomly selected by MR-GEN* for participants evaluation. During the stage of input relation definition, candidate pairs formed by IO-CTFs within candidate groups were also randomly selected. The reason we made this setup is that impact of particular sequence of candidate pairs or candidate groups can be minimized.

C. Experimental preparation and metamorphic relation identification

1) Participants and tutorial: The experiment involved 8 participants and 1 tutorial. All the participants were postgraduates with basic software testing knowledge. These participants were randomly classified into four groups: Group A, Group B, Group C, and Group D. Each group has two participants. One individual researcher with essential knowledge on metamorphic testing, category-choice frame work, METRIC, and METRIC* was appointed to be the tutorial. The role of this tutorial was to teach essential knowledge and instructions for the supporting tool to participants.

TABLE V: Specifications used in experimental stage 1

Group	Specifications
A	S_{PBC}, S_{ABBS}
B	S_{EXP}, S_{MOS}
C	S_{ABBS}, S_{PBC}
D	S_{MOS}, S_{EXP}

2) Experiment Procedure: The experiment consists of four parts including tutorial stage 1, experimental stage1, tutorial stage 2, and experimental stage 2. The details of these three stages will be illustrated below.

Tutorial Stage 1: In this stage, basic knowledge of metamorphic testing was firstly introduced to participants by the tutorial. After participants understood the principle of metamorphic testing, the tutorial introduced category-choice framework participants and one example (Credit Card System) described in Section was introduced to illustrate the concepts of categories and choices. Then METRIC and its supporting tool MR-GEN were provided to participants. Also, Credit Card System was used to illustrate the basic steps of METRIC and operations of MR-GEN. To examine participants understanding of methodology and tool, a small example was appointed to participants for exercise. During this time, participants can exchange ideas with other participants or tutorial on methodology or tool. Tutorial stage 1 lasted for one and a half hours.

As soon as tutorial stage ended. Experimental stage1 started.

Experimental Stage 1: In this stage, software specifications were assigned to participants for metamorphic relation identification using METRIC. Specifications related to each participant group are as follows:

Specifications assigned to participants within each participant group were the same, but specifications assigned to each participant group were different. Participants within a group were organized to handle specifications in the given order, which is shown in the above chart. For example, participants in Group A should firstly identify metamorphic relations in SI and then identify metamorphic relations in SII. Communications among participants were prohibited during this stage. Each participant must independently identify metamorphic relations.

As mentioned above, candidate pairs were randomly selected by MR-GEN and then provided to a participant one by one. **Each participant independently identified 40 metamorphic relations for each specification using MR-GEN.** MR-GEN is able to record the defined metamorphic relation and time spent by participants for each candidate pair. The experimental data was used to evaluate the effectiveness and efficiency of METRIC.

Tutorial Stage 2: In this stage, METRIC* and its supporting tool MR-GEN* were provided to participants. Tutorial introduced the principle of METRIC* and instructions of MR-GEN*. Credit Card System, which served as an example in Tutorial Stage 1, was also used to illustrate the basic steps of METRIC* and operations of MR-GEN*. A small example was appointed to participants for exercise to examine participants understanding of METRIC* and MR-GEN*. Participants were able to communicate with each other

TABLE VI: Specifications used in experimental stage 2

Group	Specifications
A	S_{EXP}, S_{MOS}
B	S_{PBC}, S_{ABBS}
C	S_{MOS}, S_{EXP}
D	S_{ABBS}, S_{PBC}

or tutorial on methodology or tool. Tutorial Stage 2 lasted for half an hour.

Experimental Stage 2: This stage was conducted after Tutorial Stage 2. **In this stage, specifications were assigned to participants for metamorphic relation identification using METRIC*.** Specifications related to each participant group are as follows:

Participants within a group were organized to handle specifications in the given order, which is shown in the above chart. The same as Experimental Stage 1, communications among participants were prohibited during this stage. Each participant must independently identify metamorphic relations. As mentioned above, candidate groups and candidate pairs are randomly selected by MR-GEN* to minimize the impact of a certain sequence of candidate pairs or candidate groups. **Each participant independently identified 40 metamorphic relations for each specification using MR-GEN*.** Also, the defined metamorphic relation and time spent by participants for each candidate pair are record by MR-GEN*.

D. Subject Programs and Mutant Generation

According to these specifications ,we have implemented Phone Bill Calculation(PBC), Baggage Billing Service(BBS), Car and Expense Claim System(CAR), and Meal Ordering System(MOS) respectively using Java programing language.

Phone bill calculation service is a mobile phone charge calculation system used in China Unicom. The purpose of phone bill calculation service is to determine a users phone charge within a month in terms of various kinds of aspects such as communication time, data usage, mobile phone tariffs. A user can calculate phone charge through the system.

Baggage billing service provide a passenger with baggage fee calculation service, with reference to China Airlines baggage accounting standards. A passenger can calculate his/her own baggage fees by offering relevant flight and baggage information including aircraft cabin, region, baggage weight, air fare and whether the passenger is a student to the service.

Car and expense claim system is an expense reimbursement system used in a large trading firm. CAR assists the sales director of the firm in determining the fee to be charged to each senior sales manager or sales manager for any "excessive" mileage in the use of the company car, and in processing reimbursement requests regarding various kinds of expenses such as airfare, hotel accommodation, meals, and phone calls. A sales staff uses CAR to make his claim at the end of each month.

Meal ordering system is used by an airline catering company to determine the quantity for every type of meal and other special requests (if any) that need to be prepared and loaded onto the aircraft served by the company. For each flight, MOS produces an output called "Meal Schedule Report" (MSR), which contains the following information: number of first-class meals, number of business-class meals, number of economy-class meals, number of meals for crew members, number of meals for pilots, number of child meals, and number of bundles of flowers.

Mutants for these four subject programs were generated using muJava [24]. Only method-level mutation operators were used to generating mutants, each of which contains only one fault. We generate 210, 187, 180, and 224 mutants for the four subject programs, respectively. Equivalent mutants are identified manually.

Table below provides a summary of specifications, subject programs and mutants.

E. Test Case Generation

In this paper, the source test case and follow-up test case corresponding to a metamorphic relation were generated according to IO-CTFs from which this metamorphic relation is derived. Through deciding a candidate pair formed by two IO-CTFs, sub-relation on input and sub-relation on output are determined, from which metamorphic relation is constructed. So, the application domain of a metamorphic relation identified through METRIC* is only the input and output field represented by these two IO-CTFs. When generating test cases for a metamorphic relation, it is necessary to select actual values in the input field to which the metamorphic relation applies.

For an IO-CTF, the input consists of multiple choices that belong to different categories. When generating a test case according an IO-CTF, it is necessary to take an value within the range corresponding to each choice that makes up the IO-CTF, then combine those values and form the test case. For a metamorphic relation, the generation of its corresponding test cases requires the generation of IO-CTF test cases from which this metamorphic relation is derived. One as source test case and the other as follow-up test case. These two test cases are put together as a test case for a metamorphic relation.

When selecting values from the range corresponding to a choice, we randomly generate values and select boundary values. Boundary value selection is only applied in source test case generation because if boundary value is selected in source test case generation and follow-up test case generation, there will be a combination explosion problem, resulting in excessive number of test cases.

The number of test cases generated for subject programs are shown below:

F. Measurements

Both MR-GEN and MR-GEN* can record identified metamorphic relation and time spent by participants for each candidate pair, which makes it possible to evaluate the performance of participants in terms of correctness and time taken.

If the metamorphic relation identified from a candidate pair correctly reflects the relationship among input and output of this candidate pair, then it is called a correct MR, otherwise it is an incorrect MR.

TABLE VII: Summary of specifications, subject programs and mutants

Specification	Corresponding Subject Program	LOC	Mutation Operator Used	Number of Generated Mutants	Number of Equivalent Mutants
S_{PBC}	PBC	107	AORB, AOIU, AOIS, ROR, COR, COI, LOI, SDL, VDL, ODL	210	38
S_{BBS}	BBS	97	AORB, AOIU, AOIS, ROR, COI, LOI, SDL, VDL, CDL, ODL	187	67
S_{CAR}	CAR	97	AORB, AOIU, AOIS, AOUD, ROR, COI, SDL, ODL	180	29
S_{MOS}	MOS	150	AORB, AOIU, AOIS, LOI, SDL, CDL, ODL	224	51

TABLE VIII: Number of Test Cases

Subject Program	Random			Boundary Value
	1 per MR	5 per MR	10 per MR	
PBC	142	710	1420	1096
BBS	735	3675	7350	3007
CAR	1130	5650	11300	7772
MOS	3512	17560	35120	50907

TABLE IX: Measures for participants performance

Notation	Meaning
Cmr	The proportion of correct metamorphic relations among all metamorphic relations identified by a participant
T	Time taken(in minutes) for identifying 40 MRs by a participant
Tg	Average time spent for identifying a correct metamorphic relation by a participant

As for a specification, we use **correctness(Cmr)** to evaluate the proportion of the correct metamorphic relations among all metamorphic relations identified by a participant.

Correctness was calculated as the ratio of correct metamorphic relations to all metamorphic relations (which is 40). As a reminder, metamorphic relations identified by participants were evaluated by the tutorial for several times. We use **time taken (T)** to evaluate the time spent for identifying a certain number of MRs by a participant. Therefore, time taken can be used to measure the speed of metamorphic relation identification. We use **average time taken (Tg)** to evaluate the average time spent for identifying a correct metamorphic relation by a participant. Average time taken was calculated as the ratio of time taken to the number of correct metamorphic relations.

It is obvious that higher value of correctness means better results. As for time taken and average time taken, lower value means that participants take less time to identify metamorphic relations. Therefor, lower value of time taken and average time taken mean better results.

VI. Experimental Results and Discussions

A. Feasibility of METRIC*

To answer RQ1, metamorphic relations identified by participants were counted and verified. Results are shown in the table below.

The results have shown that participants supported by METRIC* can successfully identify metamorphic relations from all these specification, thus METRIC* can be practically used for MR identification.

TABLE X: Number of identified metamorphic relations

Specification	Participant	Number of correctly identified metamorphic relations
S_{PBC}	P5	40
	P6	37
	P7	40
	P8	40
	Mean	**39.25**
S_{ABBS}	P5	40
	P6	36
	P7	40
	P8	40
	Mean	**39**
S_{EXP}	P1	40
	P2	39
	P3	32
	P4	40
	Mean	**37.75**
S_{MOS}	P1	32
	P2	35
	P3	24
	P4	40
	Mean	**32.75**

One further observation is that participants supported by METRIC* identified some metamorphic relations that are very similar to some metamorphic relations identified by participants who were supported by METRIC. The reason for this observation is as follow: the choice combinations of complete test frames involved in METRIC are same as the I-choice combinations of IO-CTFs involved in METRIC*. Output is the only difference between IO-CTF and complete test frame. Suppose we have two complete test frames (denoted as CF1 and CF2) and their corresponding IO-CTFs (denoted as IO-CTF1 and IO-CTF2). CF1 and CF2 compose a candidate pair CP1 while IO-CTF1 and IO-CTF2 compose a candidate pair CP2. As the input variation of CP1 and CP2 are the same, then the metamorphic relations within CP1 and CP2 should be same. O-choices help us to determine output relations in a more direct way. But this is not always right, for a input choice combination may relate to more than one output choice combinations. Under this circumstance, the above conclusion may not be right.

B. Difference between METRIC* and METRIC

We summarized the data of different participant groups on different experimental subjects. The results are shown in Table 1 and Table 2 below. To answer RQ1, it is necessary to compare the performance of participants supported by MR-GEN with performance of participants supported by MR-

GEN*. Comparison will be carried out from two aspects including correctness and time taken.

Table 1 summarized the correctness of different participants on different methodologies. For subject specification SPBC and SABBS, participant P1, P2, P3, and P4 were supported by METRIC when identifying metamorphic relations; participant P5, P6, P7, and P8 were supported by METRIC* when identifying metamorphic relations. Therefore, the difference in effectiveness between METRIC and METRIC* could be evaluated by comparing the performance of P1, P2, P3, and P4 with the performance of P5, P6, P7, and P8. As for subject specification SEXP and SMOS, each participant was supported by the other methodology when identifying metamorphic relations. Thus, the difference in effectiveness between METRIC and METRIC* could be evaluated by comparing the performance of participant P5, P6, P7, and P8 with those of P1, P2, P3, and P4.

From Table 1, we can observe that participants supported by METRIC* have a higher average correctness than participants supported by METRIC for subject specification SPBC and SABBS; for subject specification SEXP and SMOS, the results are the same. This result indicates that METRIC* enables software testers to identify metamorphic relations with higher accuracy, which means that METRIC* has an improvement compared with METRIC in terms of correctness. So why is there such a result? We think that this is related to the characteristics of METRIC*. When METRIC is used to support MR identification, a user needs to consider the input variation as well as output variation caused by input variation. This is not an easy task for the users when input variation is much more complicated. When it comes to METRIC*, the task of identifying metamorphic relations is decomposed into two parts. One part is to identify the relation among inputs and the other part is to identify the relation among outputs. Users do not need to consider both input and output at the same time. They only need to concentrate on judging the variation of input or the variation of output at one moment. The advantage is that it greatly reduces the difficulty of metamorphic relation identification and improves the correctness of metamorphic relation identification.

Table 2 summarizes the time taken (in seconds) of different participants on different methodologies. For subject specification SPBC and SABBS, we compared the performance of participant P1, P2, P3, and P4 with the performance of P5, P6, P7, and P8 in order to evaluate the difference in speed of identification between METRIC and METRIC*. As for subject specification SEXP and SMOS, each participant used the other methodology. Therefore, we compared the performance of participant P5, P6, P7, and P8, with those of P1, P2, P3, and P4 to evaluate the difference in speed of identification between METRIC and METRIC*.

From Table 2, it is observed that the average time taken of participants using METRIC* was lower than that of participants using METRIC. As for subject specification SEXP and SMOS, the results are the same. In addition, compared to specification SPBC, SABBS, and SMOS, the decrease of time taken for SEXP is much more significant. This result indicates that participants supported by METRIC* took fewer time in

TABLE XI: Correctness of MR Identification

Specification	METRIC		METRIC*	
	Participant	correctness	Participant	correctness
S_{PBC}	P1	0.5	P5	1
	P2	1	P6	0.925
	P3	0.975	P7	1
	P4	1	P8	1
	Mean	**0.86875**	**Mean**	**0.98125**
S_{ABBS}	P1	0.875	P5	1
	P2	0.975	P6	0.9
	P3	0.85	P7	1
	P4	0.975	P8	1
	Mean	**0.91875**	**Mean**	**0.975**
S_{EXP}	P5	0.575	P1	1
	P6	0.625	P2	0.975
	P7	0.95	P3	0.8
	P8	0.925	P4	1
	Mean	**0.76875**	**Mean**	**0.94375**
S_{MOS}	P5	0.6	P1	0.8
	P6	0.525	P2	0.875
	P7	0.85	P3	0.6
	P8	0.75	P4	1
	Mean	**0.91875**	**Mean**	**0.975**

metamorphic relation identification. Thus, METRIC* enables users to identify metamorphic relations faster and METRIC* has an improvement in terms of time taken. We believe that the reason is related to the advantage of METRIC*. It is illustrated that METRIC* can help users identify unusable candidate pairs in advance. In output relation identification stage, METRIC* enable users to find usable candidate pairs and abandon those unusable candidate pairs. Therefore, in input relation definition stage, all candidate pairs judged by the users are usable candidate pairs. METRIC does not allow a user to find unusable candidate pairs in advance. Thus, a user may encounter some unusable candidate pairs while judging candidate pairs. Since it is not able to derive metamorphic relations from unusable candidate pairs, judgement of unusable candidate pairs is a waste of time. To a certain extent, candidate pairs are screened by METRIC*. Usable candidate pairs are selected and provided to the user, eliminating the need for judging those unusable candidate pairs. As a result, METRIC* enables users to identify metamorphic relations faster.

From the above two tables, conclusions can be summarized as follows: The first is that participants supported by METRIC* have higher correctness than those supported by METRIC. This means that users make fewer mistakes when they are supported by METRIC*. The second is that participants supported by METRIC* were faster when identifying metamorphic relations. This means that users supported by METRIC* are able to get metamorphic relations more quickly. **From the above two conclusions, we can conclude that METRIC* significantly improved the identification of (correct) metamorphic relations in terms of correctness and time taken.**

MR-GEN and MR-GEN* recorded the time spent on each candidate pair for a user. Thus it is possible to **analyze changes in the speed of identifying a correct metamorphic relation from a candidate pair**. For a participant, we obtained the first twenty MRs which are correctly identified and the time spent on each MR. The reason we choose twenty is

TABLE XII: Average Time of MR Identification

Specification	METRIC		METRIC*	
	Participant	Average Time	Participant	Average Time
S_{PBC}	P1	0:02:20	P5	0:00:52
	P2	0:01:39	P6	0:01:20
	P3	0:00:37	P7	0:00:45
	P4	0:00:49	P8	0:00:49
	Mean	**0:01:21**	**Mean**	**0:00:56**
S_{ABBS}	P1	0:01:14	P5	0:00:38
	P2	0:00:56	P6	0:01:09
	P3	0:01:06	P7	0:00:53
	P4	0:00:53	P8	0:01:03
	Mean	**0:01:02**	**Mean**	**0:00:56**
S_{EXP}	P5	0:03:17	P1	0:01:01
	P6	0:03:37	P2	0:00:37
	P7	0:01:14	P3	0:00:40
	P8	0:01:33	P4	0:00:49
	Mean	**0:02:25**	**Mean**	**0:00:47**
S_{MOS}	P5	0:03:01	P1	0:02:12
	P6	0:03:49	P2	0:01:30
	P7	0:01:17	P3	0:02:01
	P8	0:02:34	P4	0:02:23
	Mean	**0:02:40**	**Mean**	**0:02:02**

due to two observations. One observation is that participant with the lowest correctness only identified 23 correct metamorphic relations. The other observation is that time spent on each metamorphic relation vary little after a participant has identified ten metamorphic relations. Average time it takes for the participants to identify the i th (i = 1, 2, , 19, 20) correct metamorphic relation is then calculated. We analyzed the variation of average time as the number of correct metamorphic relations increased to evaluate the change of participants behaviors.

For participants supported by METRIC, the average time it takes to identify the i th correct metamorphic relation is calculated as the average of all participants time spent on the i th metamorphic relation. For participants supported by METRIC*, the calculation of average time spent on i th correct metamorphic relation is more complicated. As the task of identifying metamorphic relations is decomposed into two parts in METRIC*, thus the time spent on i th metamorphic relation consists of two parts: time spent on defining input relation (tin) and time spent on defining output relation (tout). Suppose a metamorphic relation denoted as MRx is derived from candidate pair CPx which belongs to candidate group CGx. MR-GEN* recorded the time (t1) it takes for a participant to define input relation among I-choices of CPx. The tin of MRx equals to t1. MR-GEN* also records the time spent on defining output relation among output of a candidate group. Suppose CGx contains n candidate pairs and a participant takes t2 to define the output relation among O-choices of CGx, tout is calculated as the ratio of t2 to n. The reason why tout is calculated in this way is that when we define the output relation for a candidate group, we also define the output relation for all candidate pairs within this candidate group. The sum of tin and tout formed the time spent on MRx.

The four line charts above show the variation of average time spent on identifying a correct MR as the number of correct metamorphic relations increased. In these figures, the scales on X-axis represent the i th correctly defined MR and

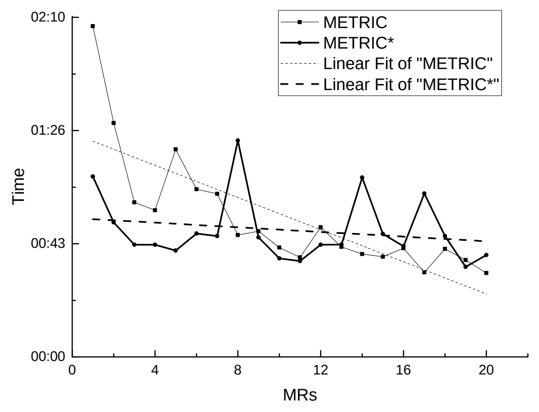


Fig. 7: Average time of the first 20 MRs(S_{PBC})

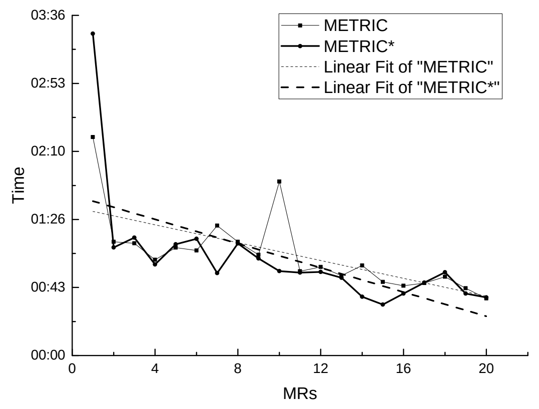


Fig. 8: Average time of the first 20 MRs(S_{ABBS})

the scales on Y-axis represent time spent on identification. Curve consisting of thin black line and black square dots represents the time cost of participants supported by METRIC. The curve consisting of thick black line and black dots represents the time cost of participants supported by METRIC*. The thin dashed line is a liner fit of all black square dots and the thick dotted line is a linear fit of all black dots.

From these figures we can observe that:

No matter of what kind of methodology, the average time

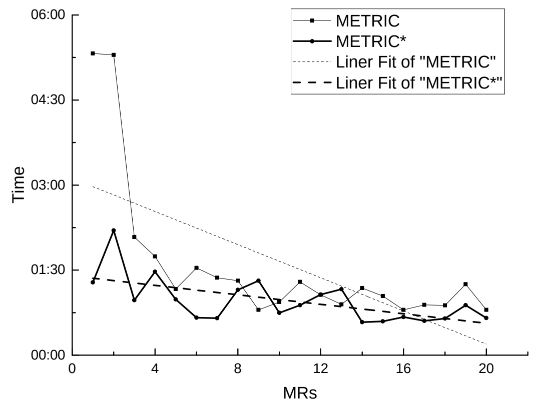


Fig. 9: Average time of the first 20 MRs(S_{EXP})

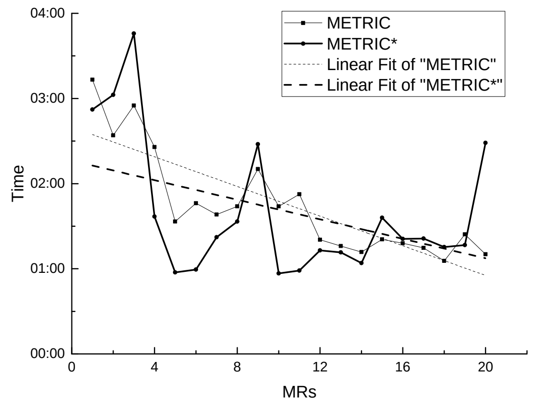


Fig. 10: Average time of the first 20 MRs(S_{MOS})

spent on identifying a correct MR from a candidate pair is decreasing. Moreover, the participants took more time to identify the first two metamorphic relations and took less time to identify later metamorphic relations. **This observation means that with the increase of correctly defined metamorphic relations, the average time cost becomes shorter, indicating that participants are getting faster and faster in a determining metamorphic relation from a candidate pair.** At first, participants had little knowledge on useful candidate pairs, so they may spent more time on determining whether a candidate pair is useful. After identifying a few metamorphic relations, participants understanding of specification is getting more and more profound and they summarized characters of useful candidate pairs. Thus, they can judge candidate pairs more quickly.

For subject specification SPBC, SEXP, and SMOS. The liner fit of METRIC curve is steeper that the liner fit of METRIC* curve, indicating that the slope of former liner fit is less than that of latter liner fit. **This observation means that the average time cost of participants supported by METRIC decreased faster than that of participants supported by METRIC*, indicating that the acceleration of former is higher.** The result of subject specification SABBS is different from those of SPBC, SEXP, and SMOS. The liner fit of METRIC* curve almost coincides with the liner fit of METRIC curve, with the slope of latter being only a litter smaller than that of former. We believe that the advent of this special case is due to special value. In result graph of SABBS, the average time of identifying first correct metamorphic relation was 0:03:24. This value is derived from the time average of four participants. Participant P2 takes 0:07:06 to judge candidate pair and define metamorphic relation while the other three participants takes 0:01:27, 0:01:47, and 0:03:18 separately. As for the latter nineteen metamorphic relations, time cost of P2 is similar to other participants time cost. Therefore, we believe that special value has an impact on the result. We think that the reasons for above result are as follows: when identifying metamorphic relations with the supported of METRIC, participants had to consider both input and output at the same time and derive output relations according to input variation, which is difficult when input variation is complicated. Before the

TABLE XIII: Number of candidate pairs and test frame group pairs

Specification	Number of test frame group pairs			Number of candidate pairs	
	Usable	Unusable	Total	Num*	Num
S_{PBC}	2+36	10+30	12+66	142	496
S_{BBS}	1+1	1+0	2+1	735	780
S_{CAR}	1+4	7+24	8+28	1130	2145
S_{MOS}	0+3512	0+12598	0+16110	3512	16110

participants had gained knowledge on useable candidate pairs, they spent a long time on judging a candidate pair and defining metamorphic relations. When several metamorphic relations were identified, they gradually accumulated the characteristics of usable candidate pairs and naturally the speed of identifying metamorphic relations were faster. The difference in speed between beginning and end is significant. As for participants supported by METRIC*, they didnt need to consider both input and output. The O-choice combinations were able to assist participants in determining output relations, which is easier than directly derive output relations according to input variation. Under these circumstances, participants supported by METRIC* were able to judge candidate pairs and define relations at a fast rate from the beginning. The difference in speed between beginning and end is less significant and acceleration is not obvious.

C. Reduction of candidate pairs

RQ3 was answered by comparing “Num *” with “Num”. If the former is less than the latter, then METRIC* can successfully reduce the number of candidate pairs. If not, then the result needs further study. Results are shown in the table below.

In the above table and charts, the column with the heading “Number of candidate pairs” show the relevant statistics. For the four specifications, the numbers in column “Num*” are less than the corresponding number in column “Num”. As for SPBC, number of candidate pairs is reduced to 142, less than one-third of the original. As for SBBS, the reduced amount of candidate pairs is small. As for SCAR, the number of candidate pairs is reduced to only half of the original. As for SMOS, the amount of reduced candidate pair is significant. **Results indicate that METRIC* can effectively identify and abandon unusable candidate pairs in advance and thus reduce the numbers of candidate pairs that need to be decided by software testers.**

One observation is that the cost of using METRIC* might be much more than using METRIC when identifying metamorphic relations from SMOS. The reason for this phenomenon is that there is only one IO-CTF in each test frame group. As shown in the figure below, if each test frame group has only one IO-CTF, then its not able to perform step 3 in METRIC* which is to identify metamorphic relations within each test frame group. Under this circumstance, identifying metamorphic relations between two distinct test frame groups is equivalent to determining metamorphic relations within a candidate pair using METRIC. The reason is that IO-CTFs within these two test frame groups can only form

TABLE XIV: Average Mutation Score

Target Program	Average Mutation Score			
	Random			Boundary
	1 per MR	5 per MR	10 per MR	
PBC	69.30%	74.71%	76.63%	80.70%
BBS	91.67%	91.67%	91.67%	91.67%
CAR	75.76%	76.55%	77.22%	86.09%
MOS	74.57%	74.57%	74.57%	74.57%

one candidate pair. If we apply METRIC* on this situation, output relation is first determined and then input relation is determined. Finally, a metamorphic relation is defined as input relation plus output relation. If we apply METRIC on this situation, the changes of input are considered and metamorphic relation is derived according to specifications. The former approach seems to be more complicated than the letter approach. Thus, when test frame groups contain a great number of groups which has only one IO-CTF, then it is preferred to use METRIC rather than METRIC*. When most of the test frame groups contain more than one IO-CTF, then METRIC* is preferred.

D. Fault detection capability

We conducted this experiment 10 times to reduce the impact of small probability events, making the results more convincing and reliable. The average mutation scores of test suites are shown in the table below. Note that column named as "Random" indicates that test suites corresponding to this column are generated randomly. "Boundary" indicates that test suites corresponding to this column are generated according to boundary conditions. "1 per MR", "5 per MR", and "10 per MR" indicates that test suites corresponding to these columns are of different sizes. Test cases that make up the test suites of column named "1 per MR" are generated 1 per MR. The other two columns are similar.

From the above table, we can observe that **average mutation score ranges from 69.30% to 91.67%, which indicates that metamorphic relations sets identified through METRIC* can detect most of the planted faults.** This demonstrates good fault detection effectiveness in software testing. When comparing test suites of different sizes, we can observe that when the size of test suite becomes larger, the mutation score does not increase significantly. As for PBC, the mutation score only increased by 7.33%, while the size of test suite increased by 10 times of the original. As for BBS and MOS, the mutation scores of test suites with different sizes are all the same. This phenomenon indicates that **increasing the size of a test suite cannot significantly improve the fault detection capability of its corresponding metamorphic relation set.**

The distribution of mutation scores corresponding to experiments is shown below.

From these four figures, we can observe that mutations scores corresponding to each experiment are distributed in small intervals, which means that **the fault detection capability of metamorphic relation sets identified through METRIC* is relatively stable.**

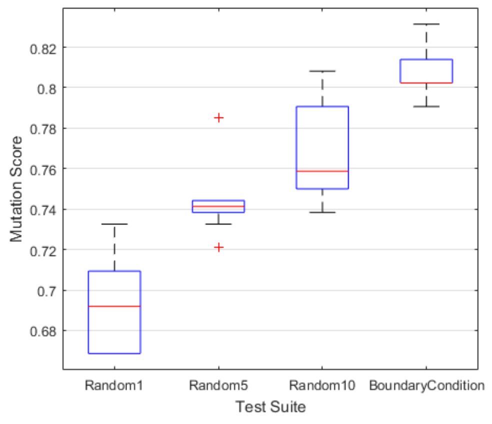


Fig. 11: Mutation score distribution of PBC

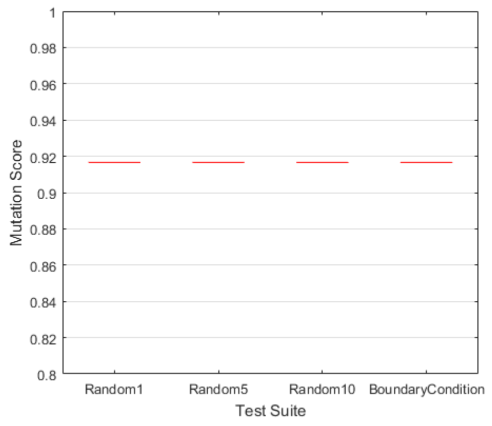


Fig. 12: Mutation score distribution of BBS

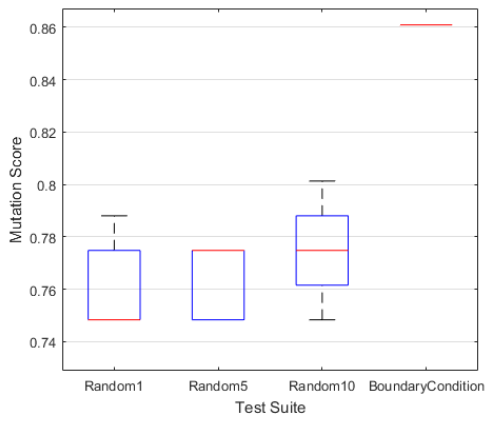


Fig. 13: Mutation score distribution of CAR

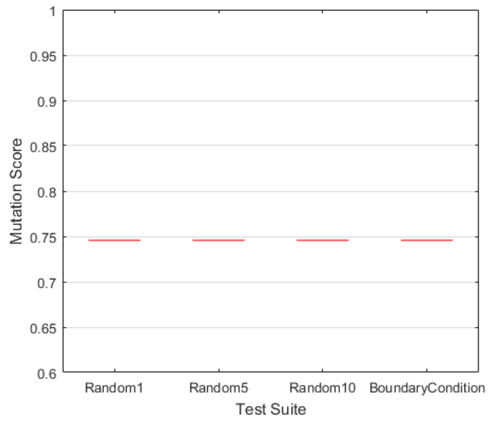


Fig. 14: Mutation score distribution of MOS

TABLE XV: Number of metamorphic relations identified by μMT and METRIC*

Subject Program	Number of MRs identified	
	Random	Boundary
PBC	32	142
BBS	36	735
CAR	60	1130
MOS	80	3152

We have also conducted experiments on studying the differences between metamorphic relations identified through METRIC* and metamorphic relations derived by other techniques in terms of fault detections capability. We select μMT as contrast technique. We evaluated the differences in fault detections capabilities by comparing the mutation scores of metamorphic sets of the same size.

We applied μMT to these four subject programs and the numbers of metamorphic relations identified are shown below.

An observation is that for these four subject programs metamorphic relations identified through μMT are less than metamorphic relations identified through METRIC*. For the sake of fairness, we randomly select some of the metamorphic relations from the all metamorphic relations of METRIC* to form a set that is the same size as the full metamorphic relation set of μMT. Suppose the subject program is PBC. When conducting an experiment, we randomly select a subset of 32 metamorphic relations from the full metamorphic relation set of METRIC*. Then we use this subset to compare with the full metamorphic relation set of μMT in terms of mutation scores. Source test cases were generated 1 per metamorphic relations randomly within the input field of a metamorphic relation, thus the test suites corresponding to these two metamorphic relation sets are of the same size. Also this experiment is repeated ten times to reduce the impact of small probability events. The distribution of mutation scores is shown below.

From above figures, we can observe that the mutation scores of metamorphic relation set corresponding to METRIC* is higher than that of metamorphic relation set corresponding to μMT, which means that **METRIC* produces a better metamorphic relation sets than the μMT produced.**

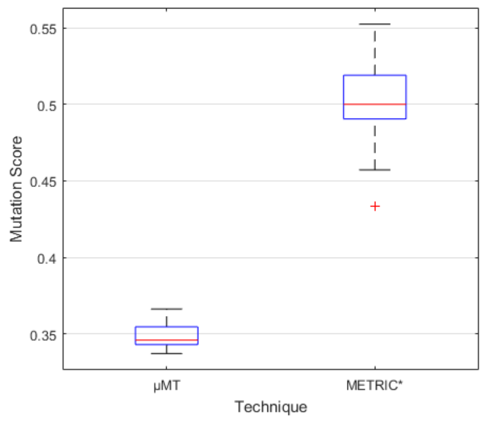


Fig. 15: Mutation score distribution of PBC

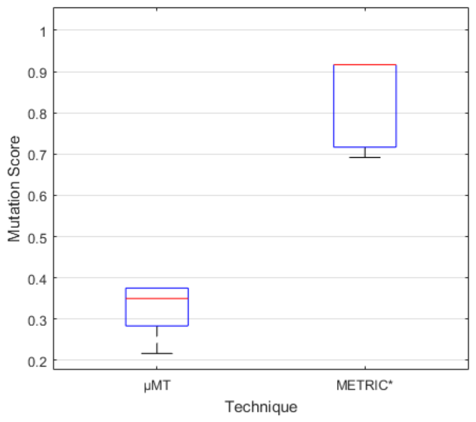


Fig. 16: Mutation score distribution of BBS

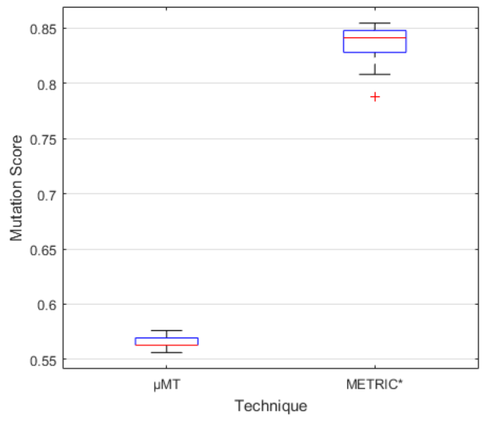


Fig. 17: Mutation score distribution of CAR

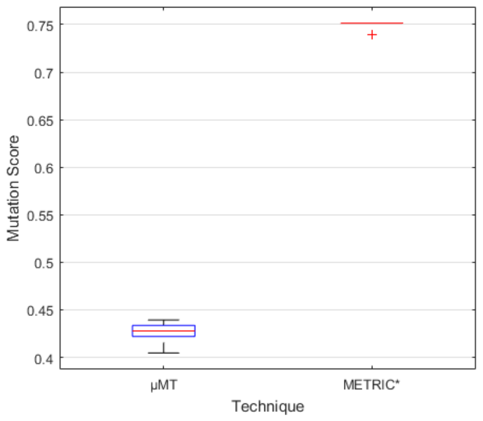


Fig. 18: Mutation score distribution of MOS

TABLE XVI: Characteristics of non-killed mutatns

Program	Mutation operators	Number of non-killed mutants	Proportion
PBC	AOIS	12	40.00%
	AORB	8	26.67%
	ODL	2	6.67%
	SDL	8	26.67%
	VDL	2	6.67%
BBS	AOIS	1	10.00%
	AORB	4	40.00%
	CDL	3	30.00%
	SDL	3	30.00%
CAR	AOIS	15	71.43%
	ROR	5	23.81%
	SDL	1	4.76%
MOS	AOIS	14	32.56%
	AORB	15	34.88%
	CDL	6	13.95%
	ODL	6	13.95%
	SDL	2	4.65%

E. Non-killed mutants

We have gathered statistics of mutants that are not killed and studied the characteristics of these mutants. Results are shown in the table below.

From the above table, it is easy to find out that **the mutants generated by mutation operator called AOIS account for the largest proportion of all non-kill mutants.** This phenomenon drives us to study the reasons behind. We will use mutant called AOIS_70 generated from program MOS to demonstrate why mutants produced by mutant operator AOIS are hard to be killed.

The fault planted in AOIS_70 is at line 91. The correct statement:

$$this.msr.numOfCMeals = this.numOfCPass * 2 \tag{1}$$

is changed into the following statement:

$$this.msr.numOfCMeals = --this.numOfCPass * 2 \tag{2}$$

The difference between these two statements is that an operator "–" is added in front of the variable named "this.numOfCPass", which let this variable be reduced by one

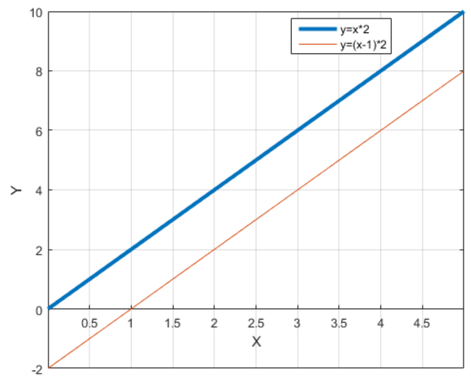


Fig. 19: Graph of two functions

before use. This statement is just before a statement which outputs the value "this.numOfCPass", so this fault can be spread out.

We treat "this.msr.numOfCMeals" as the dependent variable and denote it with y . We treat "this.numOfCPass" as independent variable and denote it with x. Before the fault is planted, the relation between independent variable and dependent variable is as follow:

$$y = x * 2 \tag{3}$$

After the fault is planted, the relation between x and y is as follow:

$$y = (x - 1) * 2 \tag{4}$$

Through the above figure we can find that the planted fault simply moves line 1 down by two unit distances. The shape of line 1 is not changed, which means that the relative relation between any two points on this line has not changed. Metamorphic testing detects faults by checking whether the relations hold between several executions. As for the variable named "numOfCPass" in this example, metamorphic testing checks whether relation between two points on this line holds. As mentioned before, relation between any two points on this line has not changed, so this mutant is hard to be killed. **Intuitively, if a fault planted cannot change the relations among output domain, then this fault is hard to be detected.**

VII. Limitations and Further Work

A. Limitations

Limitations of METRIC* can be described from two aspects. The first respect is related to characteristics of test frame groups. When most of the test frame groups contains only one IO-CTF, then METRIC* is not recommended to be applied. The reason is that identifying metamorphic relations between such test frame groups is equivalent to determining metamorphic relations within a candidate pair using METRIC. The second respect is related to the form of metamorphic relations. It is commonly known that a metamorphic relations is consist of a sub-relation on input and a sub-relation on output. But some metamorphic relations cannot be easily split into the above two parts. For example, the second metamorphic relation within Example 1 is not able to split into two parts. This metamorphic relation is described as follow: $|SP(G, A, B)| = |SP(G, A, C)| + |SP(G, C, B)|$ where C is a node in $SP(G, A, B)$, which sub-relation on input and sub-relation on output are tangled together. If most metamorphic relations are of the first form that can be split into two parts, then METRIC* can be used in most cases. Otherwise, the use of METRIC* is limited. To address this problem, it is very essential to conduct a survey on the forms of metamorphic relations.We have manually studied 153 MRs reported in [18], [9], [15], [25], [26], [27], [10], [28], [12], [29], [30], [31], [13], [11], [32], [33], [34], [14], [35], [36] and found that **14(9.15%) MRs are not able to split into input-only and output-only sub-relations.** This observation means that most MRs can be split into input-only and output-only sub-relations. Thus, METRIC* can be applied in most situations.

There are four limitations to the experiment of this study. The first limitation is that specifications involved in the experiment is simple. The number of categories identified from these specifications varies from 6 to 14 and total number of choices within these specifications varies from14 to 47. It is more appropriate to use larger specifications to verify the feasibility of METRIC*. The second limitation is that it would be better to compare the efficiency between METRIC* and METRIC. After all, these two methodology are used by software testers to identify metamorphic relations from software specifications. Although METRIC* can reduce candidate pairs theoretically, in reality, there are still many factors that will affect the procedure of metamorphic relation identification. Theoretical analysis can only reflect the degree of improvement to a certain extent. The third limitation is that categories, choices, and complete test frames used for identifying metamorphic relations are pre-defined. In reality, categories and choices identified by different software testers are different and complete test frames generated would be totally different. This phenomenon could lead to different experiment results, which is worthy of further studies. The last limitation is that the set of source test cases corresponding to μMT and METRIC* were not same, this may have impact on the results we have obtained. At first we had considered this problem and tried to construct source test cases that can be applied to metamorphic relations corresponding to these two techniques, but finally failed due to the variety of input field related to these metamorphic relations.

B. Further Work

As for the survey on the forms of metamorphic relations, we will make use of METWiki [37] to conduct the survey. METWiki is a metamorphic relations repository, which enable testers to find desired metamorphic relations for reuse or reference. It is feasible to study metamorphic relations within this repository and figure out the forms of metamorphic relations. As for the limitations to experiment, an empirical study on comparing the efficiency between METRIC* and METRIC will be conducted. Also, more complex specifications will be involved to enhance the credibility of the experiment.

VIII. Related Work

Liu et al. [18] proposed a metamorphic relation composition method which aims at constructing metamorphic relations from existing ones. If the follow-up test cases of a metamorphic relation can always be used as the source test case of another metamorphic relation, then the latter relation is "compositable" to the former one. The intuition is that when a series of various metamorphic relations are composited, the new metamorphic relation is supposed to inherit all characteristics of old metamorphic relations. An experimental study was conducted and the results showed that metamorphic relations constructed through the composition of existing metamorphic relations can effectively improve the cost-effectiveness of metamorphic testing.

Similarly, Dong et al. [36] proposed a metamorphic relation composition method based on speculative law of proposition logic. This technique constructs MRs by composing existing MRs in a pairwise way. Two case study are conducted and experimental results show that the fault detection capability of composed metamorphic relations is determined by those metamorphic relations which form the composed MR and the sequence of composition.

The application of these two techniques require existing metamorphic relations. Our work is different from the above two techniques because exiting metamorphic relations are not required and metamorphic relations are identified from specifications. This makes it possible for METRIC* to be applied in various domains and situations.

Kanewala and Bieman [38] proposed a metamorphic relation detection approach based on machine learning. This technique construct control flow graphs of functions and then extract features from control flow graphs. These features are then used to train a predictive model using machine learning algorithms. A training set is required to support the creation of predictive model. Once the training process is done, most parts of metamorphic relations detection process can be done automatically. Results of the experiment have shown that this approach is highly effective in predicting metamorphic relations under certain circumstances and predicted metamorphic relations can effectively detect faults.

Inspired by the results of the above work, Kanewala et al. [39] then proposed a machine learning approach to predict metamorphic relations in scientific software based on graph kernels. The idea is that high-level properties of the function such as loops or types of operations is not sufficient to create machine learning models, it is important to include information about operation sequences performed in a control flow graph and to include properties of individual operations. This approach consists of training phase and testing phase. In the training phase, a graph-based representation of the functions in the training set is created. Then the graph kernel is computed to score the similarity of each pair of functions in the training set. Graph kernels can be intuitively understood as functions measuring the similarity of pairs of graphs. The

TABLE XVII: Summarize of Related Techniques

Techniques	Requirements	Degree of automation
CMR [18]	Existing MRs	Manually
CMR [36]	Existing MRs	Manually
machine learning based MR detection [38]	Existing MRs	Automatically
graph kernel based MR prediction [39]	Existing MRs	Automatically
search-based MR inference [19]	Multiple program executions	Automatically
μMT [20]	Data mutation operators and mapping rules	Human involved and tool supported
METRIC [21]	Complete Test Frames	Human involved and tool supported
METRIC*	IO-CTF	Human involved and tool supported

predictive model is created by a support vector machine using graph kernel. In the testing phase, the trained model is used to predict whether a function satisfies the considered MR. Similar to metamorphic relation composition method, these two machine learning methods require existing metamorphic relations.

Zhang et al. [19] proposed a search-based approach to automatically infer polynomial metamorphic relations of numerical input programs. This approach turns the MR inferring problem into suitable value searching problem by representing a particular class of MRs using a set of parameters. The program under test is executed multiple times. Then execution results including inputs and outputs are analyzed to support the particle swarm optimization based suitable value searching problem. Three empirical studies are conducted to evaluate the proposed approach and the results show that the inferred metamorphic relations are effective in detecting faults. In contrast to compositional approaches and machine learning approaches, this approach does not require existing metamorphic relations. Also, this is a black-box approach for relations are inferred among inputs and outputs. Similarly, METRIC* is a black-box approach and does not require initial metamorphic relations. But the difference is that METRIC* makes use of program specifications but the search-based approach makes use of multiple executions of program under test.

Sun et al. [20] proposed a data mutation directed metamorphic relation acquisition methodology called μMT. This methodology uses data mutation operators to identify DM-relations among input data and uses mapping rules of the program under test to identify the output relations of the related test cases. An empirical study involving three programs was conducted to validate the feasibility of μMT and evaluate the fault detection effectiveness of derived metamorphic relations. The results have shown that μMT was feasible to derive metamorphic relations. Acquired metamorphic relations were found to be simple and demonstrated good fault detection effectiveness. We have noticed that this is also a black-box methodology. The difference between our methodology and μMT is that out methodology is based on category and choices identified from software specifications but μMT is based on data mutation.

IX. Conclusion

In this paper, we have proposed an enhanced approach named METRIC* to identify metamorphic relations based on METRIC. We innovatively introduced software output to the original METRIC and formed a new methodology. METRIC* considers both input and output of software under test to help software testers identify metamorphic relations, which is the key part of metamorphic testing. A supporting tool named MR-GEN* is implemented, which automates parts of METRIC*. Experiments are conducted and results have shown that METRIC* is feasible for identifying metamorphic relations. Metamorphic relations identified through METRIC* have shown good fault detection capability, which implies that METRIC* is able to identify good metamorphic relations and thus can improve the effectiveness of metamorphic testing.

Acknowledgment

The authors would like to thank...

References

[1] T. Y. Chen, T. H. Tse, and Z. Q. Zhou, "Fault-based testing without the need of oracles," *Information and Software Technology*, vol. 45, no. 1, pp. 1–9, 2003.

[2] J. Callahan, F. Schneider, S. Easterbrook *et al.*, "Automated software testing using model-checking," in *Proceedings 1996 SPIN workshop*, vol. 353, 1996.

[3] D. S. Rosenblum, "A practical approach to programming with assertions," *IEEE Transactions on software engineering*, vol. 21, no. 1, pp. 19–31, 1995.

[4] L. Manolache and D. G. Kourie, "Software testing using model programs," *Software: Practice and Experience*, vol. 31, no. 13, pp. 1211–1236, 2001.

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

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

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

[8] T. Y. CHEN, F.-C. KUO, H. LIU, P. POON, D. TOWEY, T. Tse, and Z. Q. ZHOU, "Metamorphic testing: A review of challenges and opportunities," *ACM Computing Surveys*, 2017.

[9] V. Le, M. Afshari, and Z. Su, "Compiler validation via equivalence modulo inputs," in *ACM SIGPLAN Notices*, vol. 49, no. 6. ACM, 2014, pp. 216–226.

[10] T. Y. Chen, J. W. Ho, H. Liu, and X. Xie, "An innovative approach for testing bioinformatics programs using metamorphic testing," *BMC bioinformatics*, vol. 10, no. 1, p. 24, 2009.

[11] L. L. Pullum and O. Ozmen, "Early results from metamorphic testing of epidemiological models," in *BioMedical Computing (BioMedCom), 2012 ASE/IEEE International Conference On*. IEEE, 2012, pp. 62–67.

[12] W. K. Chan, T. Y. Chen, H. Lu, T. Tse, and S. S. Yau, "Integration testing of context-sensitive middleware-based applications: a metamorphic approach," *International Journal of Software Engineering and Knowledge Engineering*, vol. 16, no. 05, pp. 677–703, 2006.

[13] M. Jiang, T. Y. Chen, F.-C. Kuo, and Z. Ding, "Testing central processing unit scheduling algorithms using metamorphic testing," in *Software Engineering and Service Science (ICSESS), 2013 4th IEEE International Conference on*. IEEE, 2013, pp. 530–536.

[14] T. Y. Chen, F.-C. Kuo, W. Ma, W. Susilo, D. Towey, J. Voas, and Z. Q. Zhou, "Metamorphic testing for cybersecurity," *Computer*, vol. 49, no. 6, pp. 48–55, 2016.

[15] W. K. Chan, S. C. Cheung, and K. R. Leung, "A metamorphic testing approach for online testing of service-oriented software applications," *International Journal of Web Services Research*, vol. 4, no. 2, p. 61, 2007.

[16] C.-a. Sun, G. Wang, B. Mu, H. Liu, Z. Wang, and T. Y. Chen, "Metamorphic testing for web services: Framework and a case study," in *Web Services (ICWS), 2011 IEEE International Conference on*. IEEE, 2011, pp. 283–290.

[17] A. Groce, T. Kulesza, C. Zhang, S. Shamasunder, M. Burnett, W.-K. Wong, S. Stumpf, S. Das, A. Shinsel, F. Bice *et al.*, "You are the only possible oracle: Effective test selection for end users of interactive machine learning systems," *IEEE Transactions on Software Engineering*, vol. 40, no. 3, pp. 307–323, 2014.

[18] H. Liu, X. Liu, and T. Y. Chen, "A new method for constructing metamorphic relations," in *Quality Software (QSIC), 2012 12th International Conference on*. IEEE, 2012, pp. 59–68.

[19] J. Zhang, J. Chen, D. Hao, Y. Xiong, B. Xie, L. Zhang, and H. Mei, "Search-based inference of polynomial metamorphic relations," in *Proceedings of the 29th ACM/IEEE international conference on Automated software engineering*. ACM, 2014, pp. 701–712.

[20] C.-a. Sun, Y. Liu, Z. Wang, and W. Chan, "μmt: a data mutation directed metamorphic relation acquisition methodology," in *Proceedings of the 1st International Workshop on Metamorphic Testing*. ACM, 2016, pp. 12–18.

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

[22] T. Y. Chen, P.-L. Poon, and T. Tse, "A choice relation framework for supporting category-partition test case generation," *IEEE transactions on software engineering*, vol. 29, no. 7, pp. 577–593, 2003.

[23] H. Liu, P.-L. Poon, and T. Y. Chen, "Enhancing partition testing through output variation," in *Proceedings of the 37th International Conference on Software Engineering-Volume 2*. IEEE Press, 2015, pp. 805–806.

[24] Y.-S. Ma, J. Offutt, and Y.-R. Kwon, "Mujava: a mutation system for java," in *Proceedings of the 28th international conference on Software engineering*. ACM, 2006, pp. 827–830.

[25] T. Y. Chen, T. Tse, and Z. Zhou, "Semi-proving: an integrated method based on global symbolic evaluation and metamorphic testing," *ACM SIGSOFT Software Engineering Notes*, vol. 27, no. 4, pp. 191–195, 2002.

[26] M. Jiang, T. Y. Chen, F.-C. Kuo, D. Towey, and Z. Ding, "A metamorphic testing approach for supporting program repair without the need for a test oracle," *Journal of systems and software*, vol. 126, pp. 127–140, 2017.

[27] T. Y. Chen, T. Tse, and Z. Q. Zhou, "Semi-proving: An integrated method for program proving, testing, and debugging," *IEEE Transactions on Software Engineering*, vol. 37, no. 1, pp. 109–125, 2011.

[28] Z. Q. Zhou, S. Zhang, M. Hagenbuchner, T. Tse, F.-C. Kuo, and T. Y. Chen, "Automated functional testing of online search services," *Software Testing, Verification and Reliability*, vol. 22, no. 4, pp. 221–243, 2012.

[29] X.-l. Lu, Y.-w. Dong, and C. Luo, "Testing of component-based software: A metamorphic testing methodology," in *Ubiquitous Intelligence & Computing and 7th International Conference on Autonomic & Trusted Computing (UIC/ATC), 2010 7th International Conference on*. IEEE, 2010, pp. 272–276.

[30] X. Xie, W. E. Wong, T. Y. Chen, and B. Xu, "Spectrum-based fault localization: Testing oracles are no longer mandatory," in *Quality Software (QSIC), 2011 11th International Conference on*. IEEE, 2011, pp. 1–10.

[31] F.-C. Kuo, T. Y. Chen, and W. K. Tam, "Testing embedded software by metamorphic testing: A wireless metering system case study," in *Local Computer Networks (LCN), 2011 IEEE 36th Conference on*. IEEE, 2011, pp. 291–294.

[32] S. Segura, R. M. Hierons, D. Benavides, and A. Ruiz-Cortés, "Automated metamorphic testing on the analyses of feature models," *Information and Software Technology*, vol. 53, no. 3, pp. 245–258, 2011.

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

[34] H. Liu, I. I. Yusuf, H. W. Schmidt, and T. Y. Chen, "Metamorphic fault tolerance: An automated and systematic methodology for fault tolerance in the absence of test oracle," in *Companion Proceedings of the 36th International Conference on Software Engineering*. ACM, 2014, pp. 420–423.

[35] X. Xie, J. W. Ho, C. Murphy, G. Kaiser, B. Xu, and T. Y. Chen, "Testing and validating machine learning classifiers by metamorphic testing," *Journal of Systems and Software*, vol. 84, no. 4, pp. 544–558, 2011.

[36] G.-W. Dong, B.-W. Xu, L. Chen, C.-H. Nie, and L.-L. Wang, "Case studies on testing with compositional metamorphic relations," *Journal of Southeast University (English Edition)*, vol. 24, no. 4, pp. 437–443, 2008.

[37] X. Xie, J. Li, C. Wang, and T. Y. Chen, "Looking for an mr?: try metwiki today," in *Proceedings of the 1st International Workshop on Metamorphic Testing*. ACM, 2016, pp. 1–4.

[38] U. Kanewala and J. M. Bieman, "Using machine learning techniques to detect metamorphic relations for programs without test oracles," in *Software Reliability Engineering (ISSRE), 2013 IEEE 24th International Symposium on*. IEEE, 2013, pp. 1–10.

[39] U. Kanewala, J. M. Bieman, and A. Ben-Hur, "Predicting metamorphic relations for testing scientific software: a machine learning approach using graph kernels," *Software testing, verification and reliability*, vol. 26, no. 3, pp. 245–269, 2016.