A Cost-driven Approach for Metamorphic Testing

Jing Chen[a], Fei-Ching Kuo[b], Xiaoyuan Xie[b], Lu Wang[a]
[a] Shandong Provincial Key Laboratory of Computer Network, Shandong Computer Science Center, Jinan 250014, China
Email: {chenj, wanglu}@sdas.org
[b] Faculty of Information and Communication Technologies, Swinburne University of Technology, Hawthorn, VIC 3122, Australia
Email: {dkuo, xxie}@swin.edu.au

***Abstract*— Metamorphic testing has been applied in various systems from different domains. Many studies showed that the selection of metamorphic relations greatly affected the effectiveness of metamorphic testing. However, these studies mainly focused on the fault-detection effectiveness. They did not consider the cost that metamorphic relations involved, such as the number of test inputs. Good metamorphic relations should have high fault-detection effectiveness with a low cost. In this paper, we propose a cost-driven approach for metamorphic testing. The key idea is to design metamorphic relations sharing the same test inputs to reduce the testing cost. We conduct a case study on a bank system and compare the cost-effectiveness of metamorphic relations derived from this approach and those constructed by the conventional approach. The experimental results show that metamorphic relations derived from our approach are more cost-effective. We also find that this approach not only reduces the cost of metamorphic testing, but also helps to construct different metamorphic relations to detect different types of faults.**

***Index Terms*— software testing, metamorphic testing, metamorphic relation, cost-effectiveness**

I. Introduction

Currently, software has been extensively used in various domains of the world. Along with it, massive disasters arise due to the failure of software products. As a consequence, software quality assurance becomes more and more critical. Thus, effective testing techniques are required to assure software quality in the software development. Many techniques have been proposed to guide the process of software testing, such as the selection of test cases, test automation and so on. Among these techniques, metamorphic testing (MT) [1] provides an effective mechanism to verify software outputs. It makes use of the relations over multiple inputs and their outputs to test programs without the need of oracles. To date, the studies on metamorphic testing involve two major directions. The first one is to apply MT in various systems from different domains, such as machine learning [2], [3], bioinformatics [4], middleware-based applications [5], [6], embedded software [7], online search services [8] and web services [9], [10]. The other research direction is the integration of MT and other software testing and analysis techniques, such as fault-based testing [11], program slice [12] and symbolic execution [13].

In MT, metamorphic relations (MRs) are the fundamental part. Based on MRs, follow-up test inputs are generated, and outputs of source and follow-up test inputs are compared to decide test results. Thus, the selection of metamorphic relations has a great impact on the effectiveness of MT. Some researchers studied the selection of good metamorphic relations [14], [15]. However, these studies mainly focused on the fault-detection effectiveness of metamorphic relations. They did not take account of the cost, such as the number of test inputs involved in the selected MRs. Good metamorphic relations should not only be able to detect faults, but also be cost-effective, that is, they should have high fault-detection effectiveness with a low cost. Obviously, the more test inputs are involved in the MRs, the more time is required to set up tests and run software. Previous studies that focused only on fault-detection ability usually generated different MRs that involved different test inputs. Thus, the cost of MT is increased with the number of MRs. Actually, it is more intuitive to generate different MRs that share the same source and follow-up test inputs. With such MRs, the time of execution will not be increased with the number of MRs. As a consequence, the cost-effectiveness of MT can be enhanced. Therefore, in this paper, we will show how to design metamorphic relations sharing the same test inputs and compare the cost-effectiveness of these metamorphic relations and those constructed by the conventional approach.

The rest of this paper is organized as follows. Section II introduces the background information of metamorphic testing. Section III presents a cost-driven approach for metamorphic testing. Section IV reports a case study on a bank system. And the results are discussed in Section V. Section VI discusses the threats to validity in our study. Section VII introduces related work. Section VIII concludes this paper.

This work was supported in part by the subproject of the National High Technology Research and Development Program of China (GrantNo.2012AA011202), the Australian Research Council Linkage Grant(GrantNo.ARC LP100200208), the scholarship of Shandong Provincal Education Association for International Exchanges.

doi:10.4304/jsw.9.9.2267-2275

II. METAMORPHIC TESTING

Metamorphic testing is a property-based approach. It makes use of the properties of software under test to construct metamorphic relations and applies these metamorphic relations to verify the correctness of test outputs. Suppose I_s is one input of a system P. The corresponding output is $P(I_s)$. Given another input I_f, the corresponding output is $P(I_f)$. If there exists one relation R_I between I_s and I_f and another relation R_O between $P(I_s)$ and $P(I_f)$, such that R_O is always satisfied whenever R_I is satisfied, that is,

$$R_I(I_s, I_f) \rightarrow R_O(P(I_s), P(I_f))$$

This property is called a metamorphic relation. I_s is called source test input, while I_f is follow-up test input. Then, given an implementation $P^{'}$, if $R_I(I_s, I_f)$ does not lead to the relation $R_O(P^{'}(I_s), P^{'}(I_f))$, there must exist at least one fault in $P^{'}$.

From the above definition, we can see that a metamorphic relation mainly includes two parts.

- **Input relation.** An input relation R_I specifies how the follow-up test input is constructed from the source test input.
- **Output relation.** R_O describes the relation between the outputs of the source and follow-up test inputs when R_I is satisfied.

The general procedure of applying metamorphic testing consists of the following steps.

- **Identify properties.** Testers must identify the properties from the specification. These properties may be derived from a subsystem or the whole system.
- **Design metamorphic relations.** Based on the identified properties, different metamorphic relations are designed to detect software faults.
- **Generate test inputs.** MT involves two types of test inputs. Source test inputs can be generated by using traditional testing techniques, such as random testing and fault-based testing. Follow-up test inputs are constructed from the source test inputs, based on R_I of the MRs.
- **Execute test inputs.** All the source and follow-up test inputs are executed and the corresponding test outputs are obtained.
- **Compare test outputs.** The outputs of the source and follow-up test inputs are compared to verify whether they violate the output relation R_O of the corresponding MR. If R_O is violated, one or more faults are revealed.

One simple example that calculates the average value of a set of numbers is given to illustrate how MT works. The calculation formula $avg(x_1, x_2..., x_n) = (x_1 + x_2 + ... + x_n)/n$ is the key software property listed in the requirement specification. If we permute the order of the elements, the result should remain unchanged. We can randomly generate a set of numbers as the source test input I_s, and construct the follow-up test input I_f based on R_I (permutation of elements) of this MR. We then execute the program with the source and follow-up test inputs and compare their outputs against R_O of this MR. If their outputs violate $R_O : P^{'}(I_f) = P^{'}(I_s)$, a failure is detected.

Metamorphic testing has many advantages. First of all, it provides an effective test result verification mechanism when an oracle is unavailable. Test results can be checked using metamorphic relations instead of the oracles. Certainly, MT can also be applied when the program is free from the oracle problem. Secondly, MT is independent of any programming language. It has been widely used in various applications. Finally, MT is automatable [16]–[18]. A large number of follow-up test inputs can be automatically generated and test results can be easily compared using test scripts.

As a reminder, a challenge to the application of MT is the generation of MRs. It usually requires that the testers have good domain knowledge to get enough necessary properties, from which MRs can be defined accordingly.

III. A COST-DRIVEN APPROACH FOR METAMORPHIC TESTING

In software industry, many large systems are composed of multiple subsystems and involve many terminals and massive data, such as traffic charge system and bank transaction system. When these systems are tested, it usually takes testers a great deal of time to input test data through the screens of terminals. And testers have to wait a long time to get the response because these input data may be sent to multiple subsystems to be validated and calculated for security and accuracy. Sometimes, it may take even more than 10 minutes to get a response of a test input. Moreover, since a large number of test inputs are always required to test such systems, the test cost could be very high. For example, for some industrial systems, it may take several weeks to finish all test inputs [19]. In MT, impacts from the long execution time of each test input and the large number of total test inputs are magnified because MT requires multiple executions. Moreover, in MT, multiple MRs are usually adopted. Let us consider the following example.

$$R_I^1(I_s^1, I_f^1) \rightarrow R_O^1(P(I_s^1), P(I_f^1))$$
$$R_I^2(I_s^2, I_f^2) \rightarrow R_O^2(P(I_s^2), P(I_f^2))$$

Suppose there are two MRs adopted to test a system and the source and follow-up test inputs involved in these two MRs are different (i.e. (I_s^1, I_f^1) is different from (I_s^2, I_f^2)). Then, to finish one MT with these two MRs, we need execute four different test inputs. For the above-mentioned systems with long execution time of each test input and a large number of test inputs, the cost of testing significantly increases. Actually, in a more general case, where more MRs are usually adopted, the cost is even much higher. Therefore, there is always a desire to reduce the cost of metamorphic testing. Obviously, one straightforward method is to reuse the same test inputs for different MRs.

In this study, we propose a cost-driven method of metamorphic testing. In our method, instead of generating

MRs with no restriction, we propose to generate different MRs, which share the same source and follow-up test inputs (I_s, I_f). By using such MRs, we still can detect different faults as traditional MT, but save a lot of execution time via the reuse of the test inputs.

Here is an example of sine function, which can demonstrate that MRs sharing the same test inputs can also detect different faults. There are two basic properties for sine function as follows.

- property 1: $sin(-x) = -sin(x)$
- property 2: $-1 \leq sin(x) \leq 1$

We can design the following MRs sharing the same test inputs based on these two properties.

- MR1: $sin(x) + sin(-x) = 0$
- MR2: $sin(x) - sin(-x) \leq 2$

This way of designing MRs makes these MRs sharing the same test inputs, hence it can reduce the number of test inputs and the number of test executions. Although these MRs share the same source test input x and the follow-up test input $-x$, they are distinct MRs having different fault-detection effectiveness. For instance, there are two faulty versions of the program. One version V always returns $sin(x) + 2$ and another version $V^{'}$ always returns $2sin(x)$. With any test input x, MR1 can kill V but can never kill $V^{'}$. MR2 can never kill V, but can kill $V^{'}$ with some test inputs (e.g. $x = 50°$). As a reminder, this example may be unable to demonstrate the good cost-effectiveness of this method, since the execution time of sine function is not a problem. However, for the real-life system where the execution time is very long, such MRs can significantly save the cost of MT.

Let us illustrate the difference between our cost-driven MT and traditional MT. Without loss of generality we assume that in each MR, one metamorphic test group involves one source and one follow-up test inputs.

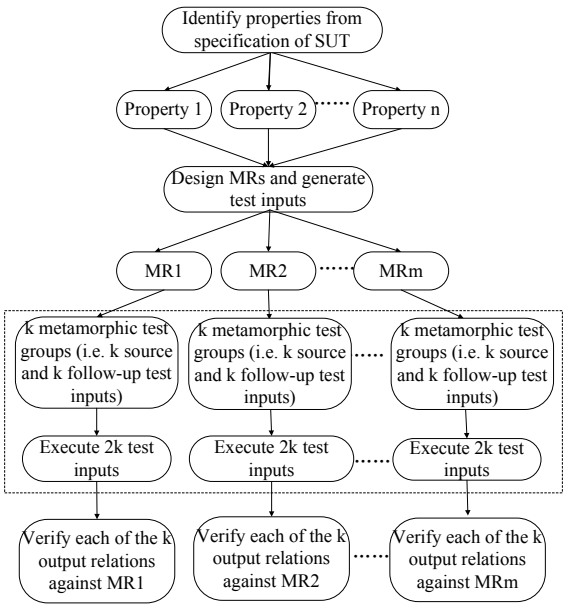

Figure 1. Traditional MT

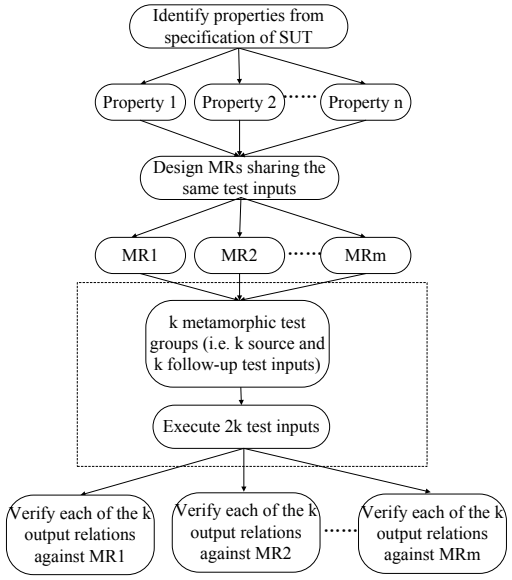

Figure 2. A cost-driven approach for MT

Figure 1 shows the process of traditional MT, where designing MRs has no restriction. Figure 2 is our cost-driven method. Suppose we have m MRs generated from n properties and for each MR, we execute k metamorphic test groups, that is, k source test inputs and k follow-up test inputs. Then, the traditional MT requires $2k * m$ executions and our method requires only $2k$ executions. It is obvious that when m and k is very large, our method can significantly save the cost of MT.

The key idea of our method is to design the MRs sharing the same test inputs. Figure 3 shows the procedure of designing the metamorphic relations sharing the same test inputs.

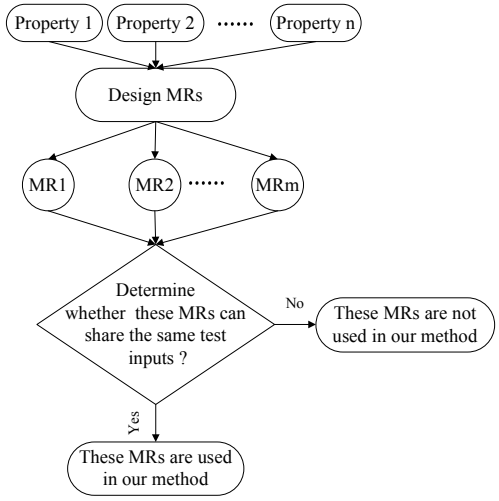

Figure 3. The procedure of designing MRs sharing the same test inputs

Based on n basic properties identified from the specification, we can design the corresponding m MRs. Then, we determine whether these MRs can share the same test inputs. If these MRs can share the same source and

follow-up test inputs, they will be used in our method. Such MRs may be derived from one, two or more properties(i.e. $n = 1, 2, ...$). As a reminder, not any arbitrary type of properties can help to derive such MRs. In the following section, we will use a case study to investigate the advantages and limitations of the approach.

IV. A Case Study

A. Test Subject: a Bank System

Figure 4 shows the process of an inter-bank transaction between two banks (the acquier and the issuer). The acquirer receives the card transaction details from the terminals, such as ATM and counter of bank, passes these data received to the issuer through an intermediate process system (CUPS). The issuer which issues this card processes the transaction data and responds to the acquirer. The program under test is a simplified bank system from the issuer. This system offers two features, such as processing inter-bank ATM withdrawal and processing inter-bank counter deposit.

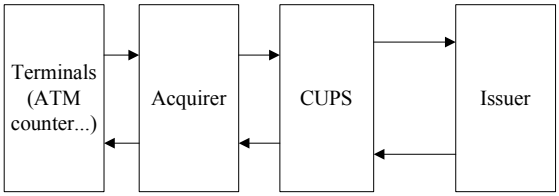

Figure 4. The process of an inter-bank transaction

The input of inter-bank ATM withdrawal is 5-tuple (CN, A, CA, CB, B), where

- CN denotes the card number.
- A denotes the transaction amount.
- CA and CB denote the city code of the acquirer and the city code of the issuer, respectively. Furthermore, $CA{=}CB$ indicates that the transaction occurs within the same city as the issuer, while $CA{\neq}CB$ indicates the transaction from different city.
- B denotes the balance of a card.

The input of inter-bank counter deposit is a 3-tuple (CN, A, B), where CN, A and B have the same meaning as those of inter-bank ATM withdrawal. The new balance NB is the output of both inter-bank ATM withdrawal and inter-bank counter deposit. According to the rules of banks, the inputs and output should satisfy the following constraints.

- As the inputs, both transaction amount and balance should be positive.
- All the inputs and output involving money, such as transaction amount, balance and new balance, should be accurate to the second decimal place. Rounding off is used in the calculation of bank transactions.
- The transaction amount should not be greater than 5000 and must be the multiple of 50 for any ATM withdrawal.
- The transaction amount can not exceed 200000 for any deposit.

TABLE I.
Transaction Fee Criterion

Transaction	Input Condition	Transaction Fee
ATM withdrawal	$CA = CB$	2
	$CA \neq CB$	$2 + 0.01A$
counter deposit	$0 < A \leq 3000$	3
	$3000 < A < 50000$	$0.001A$
	$50000 \leq A \leq 200000$	50

For inter-bank ATM withdrawal, the new balance NB is calculated based on the formula $NB = B - A - F$, where F denotes the transaction fee. For inter-bank counter deposit, the new balance NB is calculated by the formula $NB = B + A - F$. Each withdrawal or deposit can involve different transaction fee which is calculated based on different input condition shown in Table I. There are two types of transaction fees for inter-bank ATM withdrawal, which refer to the transaction within the same city and the transaction from different city, respectively. For inter-bank counter deposit, three types of transaction fees are listed according to transaction amount.

B. Metamorphic Relations of a Bank System

In order to design metamorphic relations, we firstly specify the basic properties of inter-bank ATM withdrawal as follows.

- For the transaction within the same city, the calculation of the new balance NB is based on the formula $NB = B - A - 2$.
- For the transaction from different city, the calculation of the new balance NB is based on the formula $NB = B - 1.01A - 2$.

In the same way, we can identify the basic properties of inter-bank counter deposit.

- If the transaction amount is not greater than 3000, then the calculation formula of the new balance NB should be $NB = B + A - 3$.
- If the transaction amount is not less than 50000, then the new balance NB should be calculated by the formula $NB = B + A - 50$.
- If the transaction amount is between 3000 and 50000, then we can calculate the new balance NB according to the formula $NB = B + 0.999A$.

1) Metamorphic relations constructed by the conventional approach: For mathematical functions, metamorphic relations are normally constructed based on some input transformation rules [20], such as addition and multiplication. These metamorphic relations can involve different test inputs and different test executions.

With respect to inter-bank ATM withdrawal, suppose a source test input $I_s{=}(CN, A, CA, CB, B)$. We can generally construct one follow-up test input $I_{f1}{=}(CN, K{*}A, CA, CB, K{*}B)$ by multiplying the transaction amount A and multiplying the balance B both by a positive integer K, and another follow-up test input $I_{f2}{=}(CN, A{+}C, CA, CB, B{+}C)$ by adding a positive integer C (a multiple of 50) to the transaction amount A and the balance B. If $CA{=}CB$, we can get the

following metamorphic relations based on the property $NB{=}B-A-2$.
MR1.1c: $P(I_{f1}) = K * P(I_s) + 2(K-1)$.
MR1.2c: $P(I_{f2}) = P(I_s)$.
Similarly if $CA{\neq}CB$, the following metamorphic relations can be constructed based on the property $NB{=}B-1.01A-2$.
MR2.1c: $P(I_{f1}) = K * P(I_s) + 2(K-1)$.
MR2.2c: $P(I_{f2}) = P(I_s) - 0.01C$.

In the same way, for inter-bank counter deposit, suppose a source test input $I_s{=}(CN, A, B)$. We can normally construct one follow-up test input $I_{f1}{=}(CN, K{*}A, K{*}B)$ by multiplying the transaction amount A and multiplying the balance B both by a positive integer K. Another follow-up test input $I_{f2}{=}(CN, A{+}C, B{+}C)$ can be constructed by adding a constant C to the transaction amount A and the balance B. If A, $K{*}A$ and $A{+}C$ are all within the range $(0, 3000]$, we can derive the following metamorphic relations from the property $NB = B + A - 3$.
MR3.1c: $P(I_{f1}) = K * P(I_s) + 3(K-1)$.
MR3.2c: $P(I_{f2}) = P(I_s) + 2C$.
Similarly if A, $K{*}A$ and $A{+}C$ are all within the range $[50000, 200000]$, the following metamorphic relations can be derived from the property $NB = B + A - 50$.
MR4.1c: $P(I_{f1}) = K * P(I_s) + 50(K-1)$.
MR4.2c: $P(I_{f2}) = P(I_s) + 2C$.
If A, $K{*}A$ and $A{+}C$ are all within the range $(3000, 50000)$, two metamorphic relations can be obtained based on the property $NB = B + 0.999A$.
MR5.1c: $P(I_{f1}) = K * P(I_s)$.
MR5.2c: $P(I_{f2}) = P(I_s) + 1.999C$.

Using this approach, we generate five pairs of MRs.

2) Metamorphic relations sharing the same test inputs: To demonstrate our approach, we will take MR1.1c, MR2.1c, MR3.1c, MR4.1c and MR5.1c as bases and design MRs sharing their test inputs.

For inter-bank ATM withdrawal and the property $NB{=}B{-}A{-}2$, we can design one different metamorphic relation as follows.
MR1.1s: $P(I_{f1}) = P(I_s) + (K-1) * (B-A)$.
This MR and MR1.1c share the same test inputs I_s and I_{f1} but they involve different output relations $P(I_{f1}) = P(I_s) + (K-1) * (B-A)$ and $P(I_{f1}) = K * P(I_s) + 2(K-1)$, respectively. Similarly regarding the property $NB{=}B - 1.01A - 2$, we can get the following different MR.
MR2.1s: $P(I_{f1}) = P(I_s) + (K-1) * (B-1.01A)$.

For inter-bank counter deposit and the property $NB = B + A - 3$, we can design one different metamorphic relation as follows.
MR3.1s: $P(I_{f1}) = P(I_s) + (K-1) * (B+A)$.
Accordingly, the following metamorphic relation can be derived from the property $NB = B + A - 50$.
MR4.1s: $P(I_{f1}) = P(I_s) + (K-1) * (B+A)$.
Based on the property $NB = B + 0.999A$, we can also get one different MR as follows.
MR5.1s: $P(I_{f1}) = P(I_s) + (K-1) * (B+0.999A)$.

TABLE II.
PAIRS OF METAMORPHIC RELATIONS

pair	MRs included	pair	MRs included
G_1^c	MR1.1c MR1.2c	G_1^s	MR1.1c MR1.1s
G_2^c	MR2.1c MR2.2c	G_2^s	MR2.1c MR2.1s
G_3^c	MR3.1c MR3.2c	G_3^s	MR3.1c MR3.1s
G_4^c	MR4.1c MR4.2c	G_4^s	MR4.1c MR4.1s
G_5^c	MR5.1c MR5.2c	G_5^s	MR5.1c MR5.1s

TABLE III.
NUMBER OF TEST INPUTS INVOLVED FOR EACH PAIR OF MRS

pair	number of test inputs	pair	number of test inputs
G_1^c	source: 50 follow-up: 2*50	G_1^s	source: 50 follow-up: 50
G_2^c	source: 50 follow-up: 2*50	G_2^s	source: 50 follow-up: 50
G_3^c	source: 200 follow-up: 2*200	G_3^s	source: 200 follow-up: 200
G_4^c	source: 200 follow-up: 2*200	G_4^s	source: 200 follow-up: 200
G_5^c	source: 200 follow-up: 2*200	G_5^s	source: 200 follow-up: 200

When paring MRx.1c and MRx.1s, we can obtain another five pairs of MRs for $1{\leq}x{\leq}5$, each of them sharing the same test inputs. These ten pairs of metamorphic relations are shown in Table II. G_1^c-G_5^c are composed of metamorphic relations constructed by the conventional approach, while G_1^s-G_5^s are those sharing the same test inputs. For MR1.2c, MR2.2c, MR3.2c, MR4.2c and MR5.2c, we can also design MR1.2s, MR2.2s, MR3.2s, MR4.2s and MR5.2s with our approach. However, we will only use ten pairs in Table II in the experiment. Therefore we will not list these unused MRs here.

C. Test Input Generation

We firstly use the random testing technique [21] to generate source test inputs for G_1^s-G_5^s. Considering the constraints of inputs mentioned in Section IV-A, we generate 50 and 200 source test inputs for each pair of MRs of inter-bank ATM withdrawal and inter-bank counter deposit, respectively. G_i^c uses the same source test inputs as G_i^s for $1{\leq}i{\leq}5$. Based on the source test inputs, the follow-up test inputs are constructed according to the metamorphic relations. Table III shows the number of test inputs involved for each pair of metamorphic relations. For instance, G_1^c involves 50 source test inputs and 100 follow-up test inputs, while G_1^s only involves 50 source test inputs and 50 follow-up test inputs.

D. Mutant Generation

Mutation analysis technique has been widely used to measure the effectiveness of test methods. It mainly employs some mutation operators to seed various faults into the source code of the program. A program with one or multiple faults is called a mutant. Previous research has

shown that generated mutants using mutation operators are similar to real faults [22].

We use the mujava tool [23] to automatically generate mutants. There are two levels of mutation operators for JAVA programs: method-level operators and class-level operators. The method-level operators generate the mutants at method level, while the class-level operators usually inject the structural faults at class level. In this paper, our target is to test the faults for which the program produces the incorrect outputs, such as errors in calculation, logic and condition. Hence, we select the method-level operators to generate mutants, each of which involves one single fault, either inter-method or intra-method fault. We then exclude the mutants which cause exceptions. We run these mutants with our MRs and test inputs, and analyse the data for the killable mutants (i.e. non-equivalent mutants [24]). At the end, we identify 63 and 50 killable mutants for inter-bank ATM withdrawal and inter-bank counter deposit, respectively.

E. Measurement

1) Metrics: In MT, if a mutant causes the outputs of the source and follow-up test inputs to violate a MR, we can declare that the mutant is killed by this MR and the corresponding test inputs. Researchers normally use the metric of mutation score (MS) to measure the fault-detection effectiveness of a test method. It is defined as the ratio of the number of killed mutants over the number of non-equivalent mutants.

$$MS(S,T,MRs) = \frac{N_k}{N_n}$$

where S refers to the system under test, T refers to test inputs, MRs refers to metamorphic relations, N_k refers to the number of killed mutants, N_n refers to the number of non-equivalent mutants.

Besides, we propose another metric to measure the cost-effectiveness of a test method. It is defined as the mutation score over the number of test inputs.

$$CMS(S,T,MRs) = \frac{MS}{N_T}$$

where N_T refers to the number of test inputs.

2) Imprecision: When we compare test outputs against R_O, the problem of accuracy may come due to the imprecision in floating point operations and rounding errors. The typical loss of precision in floating point operations for JAVA could cause test outputs to deviate from the expected values, even if the calculation is actually correct. It could lead to a false decision when we check whether test outputs violate a R_O with an equality. Rounding off can also cause errors. For example, for MR5.1c of inter-bank counter deposit, if we deposit 3123.52 to an account with a balance 1000.00, we will get a new balance 4120.40. If we deposit 6247.04 to an account with a balance 2000.00, the new balance should theoretically be twice as much as that of the previous deposit. However, the actual result is only 8240.79 because of the rounding error. This violate MR5.1c. We may think that the program is faulty though it is not.

Although errors due to the imprecision of floating point operations and rounding errors are not the actual faults, they can cause false positives. To address these problems, we set thresholds for the comparison of outputs in R_O. It means that test outputs are "approximately equal" and no violation of the metamorphic relation is reported if the difference between them is within the threshold.

V. Results

A. Evaluation of Cost-effectiveness

We execute all source and follow-up test inputs and verify whether their outputs violate metamorphic relations or not.

Table IV summarizes the MS and CMS of each pair of metamorphic relations. We can see that each pair of metamorphic relations derived from our approach has the same mutation score as that constructed by the conventional approach. For instance, G_1^c kills 66.67% of all mutants, and G_1^s also kills the same number of mutants. They appear to have the same fault-detection effectiveness. So do G_2^c and G_2^s, G_3^c and G_3^s, G_4^c and G_4^s, G_5^c and G_5^s. However, we also find that they have different cost-effectiveness. Each pair of metamorphic relations derived from our approach is more cost-effective than that constructed by the conventional approach. For instance, G_1^s achieves the CMS of 0.67%, while G_1^c is relatively poor only with the CMS of 0.44%. G_2^s is more cost-effective than G_2^c. So do G_3^s, G_4^s and G_5^s outperform G_3^c, G_4^c and G_5^c, respectively. It can be seen that our approach not only achieves the same fault-detection effectiveness as the conventional approach, but also reduces the cost of MT by sharing the same test inputs and test executions. Though two metamorphic relations designed by our approach share the same test inputs and test executions, they are different as discussed in Section III. We will investigate their difference in terms of fault-detection effectiveness.

TABLE IV.
MS and CMS of each pair of Metamorphic Relations

pair	MS	CMS
G_1^c	66.67%	0.44%
G_2^c	82.54%	0.55%
G_3^c	52%	0.09%
G_4^c	70%	0.12%
G_5^c	88%	0.15%
G_1^s	66.67%	0.67%
G_2^s	82.54%	0.83%
G_3^s	52%	0.13%
G_4^s	70%	0.18%
G_5^s	88%	0.22%

B. Fault-detection Effectiveness of Metamorphic Relations Sharing the Same Test Inputs

Table V summarizes the mutation score of each MR from G_1^s-G_5^s. We find that two metamorphic relations sharing the same test inputs have different fault-detection

TABLE V.
MS OF EACH MR FROM G_1^s-G_5^s

pair	MR	MS
G_1^s	MR1.1c	47.62%
	MR1.1s	39.68%
G_2^s	MR2.1c	63.49%
	MR2.1s	50.79%
G_3^s	MR3.1c	46.00%
	MR3.1s	30.00%
G_4^s	MR4.1c	64.00%
	MR4.1s	30.00%
G_5^s	MR5.1c	64.00%
	MR5.1s	52.00%

effectiveness. For instance, MR1.1c kills 47.62% of the mutants, while MR1.1s only kills 39.68% of the mutants. MR2.1c is more effective than MR2.1s in killing mutants. The same phenomenon also exists in other pairs of metamorphic relations sharing the same test inputs. We further investigate how these metamorphic relations exhibit different fault-detection capabilities. For instance, if a mutant changes the formula of inter-bank ATM withdrawal from $NB{=}B - A - 2$ to $NB{=}B + A - 2$, MR1.1s can kill this mutant but MR1.1c cannot kill it. Similarly, if another mutant changes this formula into $NB{=}B - A - 1$. Then MR1.1c can kill this mutant, but MR1.1s cannot kill it. This investigation shows that these metamorphic relations have different sensitivities to different mutants and hence exhibit different fault-detection effectiveness.

VI. THREATS TO VALIDITY

A. Internal Validity

The main threat to internal validity is the correctness of the implementation for metamorphic relations, such as test input generation, test execution and comparison of outputs. To assure the quality of the experiments, we tested the implementation thoroughly at unit level and system level.

B. External Validity

A possible threat to external validity is the representativeness of the program under test. Our program is small in size and the properties of algorithm are simple. And strictly speaking, this program does not have the oracle problem. However, we feel that the scale of program and the availability of oracle should not affect the applicability of our method. Although our case study focuses on mathematical functions, our experimental results are still meaningful. As a preliminary study, these results demonstrate the feasibility and the effectiveness of our method. In our future study, we will conduct more experimental studies which cover various types of properties.

Another threat is about mutants. Although we used the mujava tool to automatically generate mutants, the mutants can be restricted in types and be different from the actual faults. However, this method which uses mujava to generate mutants has been widely used in the literature and provided trustworthy result [22].

C. Construct Validity

The main threat to construct validity is the measurement. We use mutation score as one metric to measure the fault-detection effectiveness of metamorphic relations. It has been widely used, so this threat is acceptably alleviated. In addition, we use the ratio between mutation score and the number of test inputs as another metric to measure the cost-effectiveness of metamorphic relations. Such a metric is meaningful as this study focuses on cost of MRs.

VII. RELATED WORK

Some researchers have studied how to select good metamorphic relations. Chen et al. [14] conducted two case studies to select good metamorphic relations from black-box and white-box perspectives. They compared various metamorphic relations derived from the programs of shortest path and critical path, and attempted to distinguish more effective metamorphic relations in detecting faults. Their study showed that different MRs had different fault-detection effectiveness and only theoretical understanding of the applications was insufficient to identify good MRs. The structure of algorithm should be taken into consideration before designing metamorphic relations. Furthermore, they proposed a guideline that good metamorphic relations should be those which make the executions of the source and follow-up test inputs as different as possible. Mayer and Guderlei [15] conducted an experimental study on determinant computation to check the fault-detection effectiveness of different MRs. They not only found that the MRs with rich semantics had high fault-detection effectiveness, but also suggested based on their experiment results that the source test input and the follow-up test input of a MR should better not to execute the same part of code in order to improve the fault-detection effectiveness of this MR.

Murphy et al. [20] proposed some input transformation rules for mathematical functions, such as permutation, addition, multiplication and etc. Based on these rules, some specific metamorphic relations can be constructed. Their experiments also showed different metamorphic relations had different fault-detection effectiveness in killing mutants. Asrafi et al. [25] attempted to find the relationship between code coverage and fault-detection effectiveness. However, their studies showed high code coverage could not always guarantee high fault-detection effectiveness. Code coverage can be a good indicator for fault-detection effectiveness, but not the only one. It is necessary to consider other factors. Our experiment results show that though metamorphic relations sharing the same test inputs have the same executions, that is, they have the same code coverage, they appear different fault-detection effectiveness in killing mutants because of different output relations.

These studies above mainly focused on the fault-detection effectiveness of metamorphic relations. They did not consider the cost of test input generation and test execution. Liu et al. [26] proposed a new method

of constructing composite metamorphic relations based on original metamorphic relations. And they conducted a case study on a phylogenetic program and compared the cost-effectiveness of composite MRs and original MRs. Their study indicated that composite metamorphic relations had higher cost-effectiveness than original metamorphic relations. And their approach involved fewer metamorphic relations and test executions. In this paper, we improve the cost-effectiveness of metamorphic testing by designing MRs sharing the same test inputs and the same test executions. This is a novel approach to designing metamorphic relations.

VIII. Conclusion

Metamorphic testing is an effective property-based approach. The selection of metamorphic relations has a great impact on the effectiveness of MT. Previous studies mainly focused on fault-detection effectiveness, while the cost was seldom studied. In this paper, we propose a novel approach for metamorphic testing, which can construct different metamorphic relations sharing the same test inputs to reduce the testing cost. We conduct a case study on a bank system and compare the cost-effectiveness of metamorphic relations derived from our approach and those constructed by the conventional approach. The experimental results show that the metamorphic relations derived from our approach are more cost-effective. And more importantly, we further find that these metamorphic relations appear different fault-detection effectiveness and kill different mutants though they involve the same test inputs and test executions.

Our approach of designing metamorphic relations is suitable for all mathematical functions. For instance, we are testing a program implementing a function $f(x_1, ..., x_n)$. This function can be presented in multiple forms as follows.

$$f(x_1, ..., x_n) = f(x_1^{'}, ..., x_n^{'}) + h(x_1^{'}, ..., x_n^{'}) + c_1$$

$$f(x_1, ..., x_n) = k * f(x_1^{'}, ..., x_n^{'}) + l(x_1^{'}, ..., x_n^{'}) + c_2$$

$$f(x_1, ..., x_n) = (k-1) * f(x_1^{'}, ..., x_n^{'}) + w(x_1^{'}, ..., x_n^{'}) + c_3$$

....., etc.

Thus we can get multiple metamorphic relations sharing the same test inputs and the same test executions. However, if there exist too much input parameters and complex calculations in the equations h, l, w and etc, the output relations will become very complex and thus it is error-prone when test outputs $f(x_1, ..., x_n)$ and $f(x_1^{'}, ..., x_n^{'})$ are compared. In real life, many systems involve mathematical functions. Therefore, this approach is useful in practical software testing. We shall further study this issue for various systems and explore more complicated properties in our future work.

Acknowledgment

The authors are grateful to the anonymous referees for their valuable comments and suggestions to improve the presentation of this paper.

References

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

[2] X. Xie, J. W. K. 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, pp. 544–558, 2011.

[3] C. Murphy, G. Kaiser, L. Hu, and L. Wu, "Properties of machine learning applications for use in metamorphic testing," in *SEKE'08, Proceedings of the 20th International Conference on Software Engineering and Knowledge Engineering*, 2008, pp. 867–872.

[4] T. Y. Chen, J. W. K. Ho, H. Liu, and X. Xie, "An innovative approach for testing bioinformatics programs using metamorphic testing," *BMC Bioinformatics*, vol. 10, pp. 24–35, 2009.

[5] T. H. Tse, S. S. Yau, W. K. Chan, H. Lu, and T. Y. Chen, "Testing context-sensitive middleware-based software applications," in *COMPSAC'04, Proceedings of the 28th Annual International Conference on Computer Software and Applications*, 2004, pp. 458–466.

[6] W. K. Chan, T. Y. Chen, H. Lu, T. H. 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. 5, pp. 677–703, 2006.

[7] 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, 36th Annual IEEE Conference on*, 2011, pp. 291–294.

[8] Z. Q. Zhou, S. Zhang, M. Hagenbuchner, T. H. Tse, F.-C. Kuo, and T. Y. Chen, "Automated functional testing of online search services," *Software Testing,Verification and Reliability*, vol. 22, pp. 221–243, 2012.

[9] W. K. Chan, S. C. Cheung, and K. R. P. H. Leung, "A metamorphic testing approach for online testing of service-oriented software applications," *International Journal of Web Services Research*, vol. 4, no. 2, pp. 61–81, 2007.

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

[11] T. Y. Chen, T. H. Tse, and Z. Q. Zhou, "Fault-based testing without the need of oracles," *Information and Software Technology*, vol. 45, pp. 1–9, 2003.

[12] 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, pp. 866–879, 2013.

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

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

[15] J. Mayer and R. Guderlei, "An empirical study on the selection of good metamorphic relations," in *COMPSAC'06, Proceedings of the 30th Annual International Computer Software and Application Conference*, 2006, pp. 475–484.

[16] C. Murphy, K. Shen, and G. Kaiser, "Automatic system testing of programs without test oracles," in *ISSTA'09, Proceedings of the 2009 ACM International Symposium on Software Testing and Analysis*, 2009, pp. 189–199.

[17] A. Gotlieb and B. Botella, "Automated metamorphic testing," in *COMPSAC'03, Proceedings of the 27th Annual International Computer Software and Applications Conference*, 2003, pp. 34–40.
[18] S. Segura, R. M. Hierons, D. Benavides, and A. Ruiz-Cortes, "Automated metamorphic testing on the analyses of feature models," *Information and Software Technology*, vol. 53, pp. 245–258, 2011.
[19] G. Rothermel, R. H. Untch, C. Chu, and M. J. Harrold, "Prioritizing test cases for regression testing," *Software Engineering, IEEE Transactions on*, vol. 27, no. 10, pp. 929–948, 2001.
[20] C. Murphy, "Metamorphic testing techniques to detect defects in applications without test oracles," Department of Computer Science, Columbia University, Tech. Rep. cucs-010-10, 2010.
[21] J. W. Duran and S. C. Ntafos, "An evaluation of random testing," *Software Engineering, IEEE Transactions on*, vol. 10, no. 4, pp. 438–444, 1984.
[22] J. H. Andrews, L. C. Briand, and Y. Labiche, "Is mutation an appropriate tool for testing experiments," in *ICSE'05, Proceedings of the 27th International Conference on Software Engineering*, 2005, pp. 402–411.
[23] Y.-S. Ma, J. Offutt, and Y. R. Kwon, "Mujava: An automated class mutation system," *Journal of Software Testing, Verification and Reliability*, vol. 15, no. 2, pp. 97–133, 2005.
[24] D. Schuler and A. Zeller, "Covering and uncovering equivalent mutants," *Software Testing, Verification and Reliability*, vol. 23, pp. 353–374, 2013.
[25] M. Asrafi, H. Liu, and F.-C. Kuo, "On testing effectiveness of metamorphic relations: A case study," in *2011 Fifth International Conference on Secure Sofeware Integration and Relaibility Improvement*, 2011, pp. 147–156.
[26] H. Liu, X. Liu, and T. Y. Chen, "A new method for constructing metamorphic relations," in *QSIC'12, 2012 12th International Conference on Quality Software*, 2012, pp. 59–68.

Jing Chen received her B.S. degree in applied physics from Qingdao University, China in July 2001 and M.S. degree in optical engineering from Beijing Institute of Technology, China in March 2004. She is currently an associate professor working in Shandong Computer Science Center, China. Her current research interests focus on software testing and cloud computing.

Fei-Ching Kuo(member of IEEE Computer Society) received her Bachelor of Science Honours in Computer Science and PhD in Software Engineering, both from Swinburne University of Technology, Australia. She was a lecturer at University of Wollongong, Australia. She is currently a Senior Lecturer at Swinburne University of Technology, Australia. She is also the Program Committee Chair for the 10th International Conference on Quality Software 2010 (QSIC'10) and Guest Editor of a Special Issue for the Journal of Systems and Software, special issue for Software Practice and Experience, and special issue for International Journal of Software Engineering and Knowledge Engineering. Her current research interests include software analysis, testing and debugging.

Xiaoyuan Xie received her Bachelor and Master degrees in Computer Science from Southeast University, China in 2005 and 2007, respectively. And she received her PhD degree in Computer Science from Swinburne University of Technology, Australia in 2012. She is now working as the postdoctoral research fellow in Swinburne University of Technology. Her research interests include software analysis and testing, fault localization, debugging, search-based software engineering.

Lu Wang received his B.S. degree in computer science and technologies and M.S. degree in software engineering from Shandong University, China in 2004 and 2007, respectively. He is currently an researcher at Shandong Computer Science Center, China. His research interests include software testing and cloud computing.