

# Improving The Effectiveness of Automatically Generated Test Suites Using Metamorphic Testing

Prashanta Saha

prashantasaha@montana.edu  
School of Computing, Montana State University  
Bozeman, Montana, USA

Upulee Kanewala

upulee.kanewala@montana.edu  
School of Computing, Montana State University  
Bozeman, Montana, USA

## ABSTRACT

Automated test generation has helped to reduce the cost of software testing. However, developing effective test oracles for these automatically generated test inputs is a challenging task. Therefore, most automated test generation tools use trivial oracles that reduce the fault detection effectiveness of these automatically generated test cases. In this work, we provide results of an empirical study showing that utilizing metamorphic relations can increase the fault detection effectiveness of automatically generated test cases.

## KEYWORDS

Metamorphic testing, metamorphic relation, automated test case generation

### ACM Reference Format:

Prashanta Saha and Upulee Kanewala. 2020. Improving The Effectiveness of Automatically Generated Test Suites Using Metamorphic Testing. In *IEEE/ACM 42nd International Conference on Software Engineering Workshops (ICSEW'20)*, May 23–29, 2020, Seoul, Republic of Korea. ACM, Seoul, South Korea, 2 pages. <https://doi.org/10.1145/nnnnnnn.nnnnnnn>

## 1 INTRODUCTION

Software testing is a costly activity yet essential to detect faults. Typically in testing, an *oracle* is used to check whether the output produced for a given test input is correct or not [6]. Much work has been done on automated test case generation, including the development of publicly available tools [3]. The main focus of this work has been on developing efficient methods to generate test inputs to achieve a particular target such as coverage and weak mutation score [5]. However, there has been relatively less attention paid on utilizing effective oracles to improve the fault detection effectiveness of these automatically generated test cases.

Metamorphic Testing (MT) is a technique proposed to alleviate the oracle problem of software under test (SUT) [1]. This is based on the idea that most of the time it is easier to predict relations between outputs of a program, than understanding its input-output behavior. Such a relation is called a Metamorphic Relation (MR) in MT, and is a necessary property of the SUT that specifies a relationship between multiple inputs and their outputs [2].

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](mailto:permissions@acm.org).  
*ICSEW'20*, May 23–29, 2020, Seoul, Republic of Korea

© 2020 Association for Computing Machinery.  
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM... \$15.00  
<https://doi.org/10.1145/nnnnnnn.nnnnnnn>

Automatically generated test suites have certain advantages over manually written test cases, in particular, saving human labor and time. Some work has shown that it is more effective to use test cases that are generated based on some coverage criteria rather than randomly generated test cases [4]. However, due to the automated generation of test inputs, defining the oracles for these test inputs is a hard problem and faces the oracle problem. Thus, many of the automatically generated test cases would contain trivial oracles, such as the assert statements that we discussed above. This reduces the fault detection effectiveness of these test cases. Therefore, in this work, we investigate whether we can utilize MRs to improve the fault detection effectiveness of automatically generated test cases. For example, figure 1a is an EvoSuite generated test case for *Power* method. This method powers a matrix of the given component (i.e. *int n*) and returns the powered matrix. Though this test case has a code coverage of 100% but the generated assert statements are weak to detect critical faults in the method. Because of the presence of such trivial oracles, the fault detection effectiveness of this test case is reduced. With Multiplication MR, we modified the current test case from figure 1b. We multiplied the source test case matrix with the same matrix. We ran the test case for the *Power* method. Then we expected the resultant matrix from these two test cases are equal, or the follow-up output is higher than source output and compared them using assertion statements.

In this paper, we present the initial results of an empirical study conducted to evaluate the effectiveness of utilizing MRs with automatically generated test inputs. Our preliminary results show that MRs can help to increase the effectiveness of automatically generated test suites.

## 2 EMPIRICAL STUDY

In this experiment we used 4 classes (*Matrix.java*, *LeastSquaresSolver.java*, *ForwardBackSubstitutionSolver.java* and *SquareRootSolver.java*) from *la4j*<sup>1</sup> (version 0.6.0) open-source Java library. *la4j* is a linear algebra library that provides matrix and vector implementations and algorithms and was one of the software packages used for evaluating the performance of automated testing tools. For each of these 4 classes, we used EvoSuite [3] tool to generate test cases targeting line, branch, and weak mutation coverage. We have identified 16 MRs for the above 4 classes. These MRs are created based on common matrix operations (e.g., Transpose Matrix, Identity Matrix). We manually verified those input-output relationships of MRs with some sample values. Then we ran those MR modified source test cases (follow-up test cases) with automated source test inputs on the original programs and verified the MR properties again. If any

<sup>1</sup><http://la4j.org/>

```

(a)
@Test(timeout = 4000)
public void test042() throws Throwable {
    MockRandom mockRandom0 = new
    MockRandom();
    assertNotNull(mockRandom0);

    DenseMatrix denseMatrix0 =
    DenseMatrix.randomSymmetric(0,
    mockRandom0);
    assertEquals(0, denseMatrix0.columns());
    assertEquals(0, denseMatrix0.rows());
    assertNotNull(denseMatrix0);

    Matrix matrix0 = denseMatrix0.power(1293);
    assertNotSame(denseMatrix0, matrix0);
    assertNotSame(matrix0, denseMatrix0);
    assertEquals(0, denseMatrix0.columns());
    assertEquals(0, denseMatrix0.rows());
    assertEquals(0, matrix0.rows());
    assertEquals(0, matrix0.columns());
    assertTrue(matrix0.equals((Object)denseMatrix0));
    assertNotNull(matrix0);
}

(b)
@Test(timeout = 4000)
public void test042() throws Throwable {
    MockRandom mockRandom0 = new
    MockRandom();
    assertNotNull(mockRandom0);

    DenseMatrix denseMatrix0 =
    DenseMatrix.randomSymmetric(0,
    mockRandom0);
    assertEquals(0, denseMatrix0.columns());
    assertEquals(0, denseMatrix0.rows());
    assertNotNull(denseMatrix0);

    Matrix matrix0 = denseMatrix0.power(1293);

    //Matrix Multiplication - MR
    Matrix matrix1 =
    denseMatrix0.multiply(denseMatrix0);
    matrix1 = matrix1.power(1293);
    assertTrue(matrix0.equals((Object)matrix1
    ));
}

```

Figure 1: (a) EvoSuite Generated Test Case , (b) Modified Test Case with MR in MT

MR did not hold for any test input, we excluded that MR for that particular input.

We used mutation testing, in particular, PIT<sup>2</sup> tool to generate mutants, to measure the fault detection effectiveness of the test cases enhanced with MRs. We considered a mutant as "killed" when the MR violates the output relation and as "alive" when the relationship holds. We collected all the killed/alive information and calculated the mutation score and fault detection ratio for automated test suites and MRs.

### 3 PRELIMINARY RESULTS AND CONCLUSIONS

Figure 2 shows the fault detection effectiveness of EvoSuite generated test cases (orange), and the EvoSuite test cases utilizing MRs (blue). We also show the fault detection effectiveness of developer written test cases. As shown in the results, there is a significant increase in the mutation score of when MRs are utilized with the automatically generated test suite. For two classes, the increase of the mutation score is 100% higher than the automatically generated test suite. This preliminary result suggests that utilizing MRs with automatically generated test cases would improve the fault detection effectiveness. But for the case of the developer test suite, there is no additional mutant killed by the MRs except for Matrix. This needs to be investigated further.

Our preliminary results are promising, and it suggests that MT can effectively improve the fault detection capability of automatically generated test suites. But we need a large scale implementation to prove this claim further and to validate the correlation. We also need to find out the individual performance of MRs compared to the automatically generated test suites.

### 4 ACKNOWLEDGMENTS

This work is supported by award number 1656877 from the National Science Foundation. Any Opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect those of the National Science Foundation.

<sup>2</sup><https://pitest.org/>

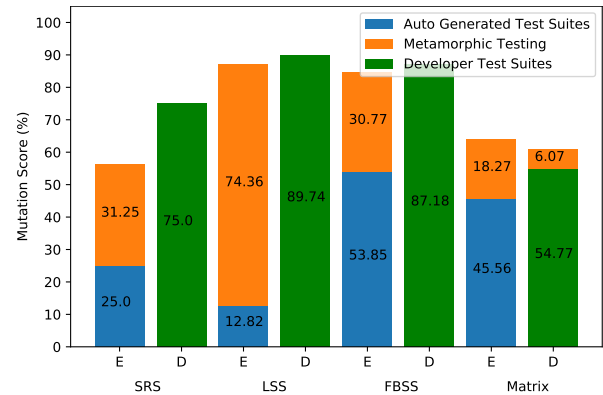


Figure 2: 4 classes with Mutation score of automatically generated test suites and developer test suites, and increase of mutation score with Metamorphic Testing. (SRS = Square-RootSolver, LSS = LeastSquaresSolver, FBSS = ForwardBack-SubstitutionSolver, E = EvoSuite, D = Developer)

### REFERENCES

- [1] Tsong Yueh Chen, S. C. Cheung, and S. W. Yiu. 1998. Metamorphic testing: a new approach for generating next test cases.
- [2] Tsong Yueh Chen, Fei-Ching Kuo, Huai Liu, Pak-Lok Poon, Dave Towey, T. H. Tse, and Zhi Quan Zhou. 2018. Metamorphic Testing: A Review of Challenges and Opportunities. *ACM Comput. Surv.* 51, 1, Article 4 (Jan. 2018), 27 pages. <https://doi.org/10.1145/3143561>
- [3] Gordon Fraser and Andrea Arcuri. 2011. EvoSuite: Automatic Test Suite Generation for Object-oriented Software (*ESEC/FSE '11*). ACM, New York, NY, USA, 416–419. <https://doi.org/10.1145/2025113.2025179>
- [4] Carlos Pacheco and Michael D. Ernst. 2007. Randoop: Feedback-directed Random Testing for Java (*OOPSLA '07*). ACM, New York, NY, USA, 815–816. <https://doi.org/10.1145/1297846.1297902>
- [5] Prashanta Saha and Upulee Kanewala. 2018. Fault Detection Effectiveness of Source Test Case Generation Strategies for Metamorphic Testing (*MET '18*). ACM, New York, NY, USA, 2–9. <https://doi.org/10.1145/3193977.3193982>
- [6] Elaine J. Weyuker. 1982. On Testing Non-Testable Programs. *Comput. J.* 25, 4 (11 1982), 465–470. <https://doi.org/10.1093/comjnl/25.4.465> arXiv:<http://oup.prod.sis.lan/comjnl/article-pdf/25/4/465/1045637/25-4-465.pdf>