2017 IEEE 28th International Symposium on Software Reliability Engineering Workshops

Identifying Failed Test Cases through Metamorphic Testing

Zhan-wei Hui[1], Song Huang[1,2]
[1]Software Testing and Evaluation Centre
PLA University of Science and Technology
Nanjing, Jiangsu Province, China
[2]PLA Military Training Software Testing and Evaluation Centre
Nanjing, Jiangsu Province, China
{hzw_1983821,hs0317}@163.com

Tsong Yueh Chen, Man F. Lau*, Sebastian Ng
Dept. of Computer Science and Software Engineering
Swinburne University of Technology
Melbourne, Victoria, Australia
{tychen,elau,sng}@swin.edu.au

***Abstract*— When testing a program without a test oracle, it is impossible to know whether a test case will lead to a failure or not. The infeasibility to identify a failed test case severely restricts the applicability of many testing, debugging and fault localization techniques. However, with the help of Metamorphic Testing (MT) which was proposed to alleviate the test oracle problem, we are able to estimate how likely a test case is a failed test case.**

Keywords- metamorphic testing; software testing; test oracle

I. INTRODUCTION

A *failed* test case is one that causes the program under test to fail. Given a program that implements an "algorithm", a *test oracle* of the program is a mechanism to verify whether the input-output (I/O) relationship of the program holds according to the "algorithm". In traditional testing, test oracle is needed to determine whether a test case is failed or not. Knowing the failed test cases is important in software development. For example, developers use the failed tests cases to perform debugging by tracing the program slices that are executed by these test cases. Another example is in fault localization, spectrum-based fault localization (SBFL) techniques make use of how frequently a statement is executed or not executed by a failed test case or a non-failed test case to "localize" a fault in a buggy program [1]. SBFL is based on two intuitions: (1) "a statement is more likely to be faulty if it is executed by more failed test cases" and (2) "a statement is less likely to be faulty if it is executed by more non-failed test cases". The likelihood of a statement being faulty can be calculated using such data.

Metamorphic Testing (MT) aims to test programs that have test oracle problems [2,3]. A program is said to have a *test oracle problem* if its test oracle does not exist or its test oracle is practically infeasible (e.g., too time-consuming) to verify the I/O relationship of the program. Assume that we have a program P that implements an algorithm and that P does not have a test oracle. After executing P with a test case t, we have an actual output o (that is, $o = P(t)$). Since P does not have a test oracle, there is no way to tell whether t and o together satisfies the I/O relationship. MT uses certain property of the algorithm to verify the correctness of the program P. Such a property is usually referred to as a *metamorphic relation* (MR). MT uses a MR, the *source* test case t (and its actual output o, if necessary) to generate a *follow-up test case* t'. After executing P with t', the actual output o' of t' is obtained (that is, $o' = P(t')$). MT then verifies whether t, o, t' and o' satisfy the relevant MR. If the MR is violated (that is, not satisfied), P can be concluded to be incorrect. Otherwise, P passes MT for this MR with respect to t and t'. This group of source and follow-up test cases is referred to as a *metamorphic test group* (MTG). When MT reveals failures in a program P, identifying the failed test case becomes an issue. Since there is no test oracle, we cannot have a definite answer about which test case is a failed test case. Should this be the source test case? Or the follow-up test case?

In this paper, we propose an approach to "determine" how likely a test case is a failed test case after MT has been performed for testing programs with test oracle problem. Our intention is to gather as much information on the likelihood of a test case being failed as possible to help in the debugging and fault localization processes.

II. PROPOSED APPROACH

In performing MT, many MRs will be used to help verify the correctness of a program. Usually, for each MR, there will be many different MTGs. After MT has been performed, we will have a record of which test cases were involved in which MRs (either as source or follow-up test cases) and a record of whether the relevant MR is satisfied (S) or violated (V) with respect to the MTGs. Assume that we have a program *PSin* that computes the sine function and further that we have a record like the one as shown in Table I after performing MT on *PSin* with 3 MRs and 2–3 MTG for each individual MR. For example, MR1 is satisfied for the MTG <20, 380> and MR3 is violated for the MTG <20, 160>. Based on the record, the *Suspiciousness* value of a test case t can be calculated by (1)

$$\text{Suspiciousness}(t) = {NV(t)}\Big/{NMR(t)} \quad (1)$$

where $NV(t)$ is the number of MRs being involved in t that is violated and $NMR(t)$ is the number of MRs being involved in t. The suspiciousness value of the test case 20 is 1/3 because $NV(20) = 1$ and $NMR(20) = 3$.

Once the suspiciousness values of all test cases are computed, test cases that have very high suspiciousness value are likely to be failed test cases. Our intuition is the more violation of the MRs involved with a particular test case, the higher is the chance of the test case being a failed

*Corresponding author.

978-1-5386-2387-9/17 $31.00 © 2017 IEEE
DOI 10.1109/ISSREW.2017.23

TABLE I. SATISFACTION OR VIOLATION OF MRS (E.G. PROGRAM COMPUTES SIN(X))

Test case	MR1 sin(x) = sin(360+x)		MR2 sin(-x) = -sin(x)			MR3 sin(x) = sin(180-x)	
20	S		S			V	
380	S	V			V		S
740		V					
-20			S				
160				V		V	
-160				V			
-200							S
-380					V		

test case. Equation (1) is based on this intuition, which is basically the ratio of violated relevant MRs and the total number of relevant MRs. This paper only reports Equation (1). In fact, there are many formulas in SBFL approaches that could be used in calculating these suspicious values. Moreover, we have investigated many formulas to calculate the likelihood for a test case to be a failed test case.

III. EXPERIMENTS

An experiment has been performed to validate our approach. Eight programs have been selected for our study, the first six programs were selected from MT's research literature and the last two programs were selected from a practical terrain calculation projects in geographic information system. Table II lists all these programs. For example, the program NBC, selected from [4], is a classic bias classification program written in Java.

For each subject program, 100 mutated versions (each having a single fault different from the original version) were randomly generated using either Milu mutation generation tool [4] for C programs or muJava mutation generation tool [5] for Java programs. Each mutated version is considered to be a faulty version of the original subject program. Our approach is to use the suspiciousness values of the test cases to "identify" the potential failed test cases for the faulty versions.

Table II also shows the number of MRs used for each program. For example, 13 MRs were used for the NBC program. We adopt random testing strategy to randomly generate 100 source test cases. We then generate the corresponding follow-up test cases according to each MR. Once all source and follow-up test cases are generated, we then perform MT on each individual MTG on each individual mutated version. We then record whether the MR is satisfied or not for each MTG for that particular mutated version.

After all test cases have been executed, the corresponding satisfaction or violation record of each version will be similar to that listed in Table I. Based on the record, we can calculate the suspiciousness value of each test case using (1) and select the test case with the highest suspiciousness value as the failed test case.

IV. PRELIMINARY RESULTS AND CONCLUSIONS

In summary, we have 8 programs in our experiment. For each program, we have one original version and 100 faulty versions. We do not assume that the original version is correct as our work assumes that we do not have a test oracle

TABLE II. SUBJECT PROGRAMS

Program	Description	Prog. Lang.	# of MRs
Sin	Calculate sine function	C	17
Trisquare	Calculate the area of a triangle	C	16
P	Calculate the definite integral of a function between a and b	C	10
NBC	Classic bias classification program based on k-Nearest Neighborhood	Java	13
BinSearch	Search an integer in an ordered data structure	C	12
TCAS	Air collision detection	C	37
Area	Area calculation in a digital map	C	9
Grade	Gradient calculation in a digital map	C	9

for the program under test. For each version, we perform MT with 100 MTGs. After analyzing all test cases returned with the highest suspiciousness values, we found it very "interesting" and "exciting" that there is a very high correlation of these test cases being the failed test cases of the faulty versions.

This preliminary result is very exciting and promising. However, we need a larger scale experimentation involving more subject programs that are more complex to further validate the observed correlation. More experiments would be designed and carried out to investigate the issues further. Apart from Equation (1), we anticipate to use some of the popular risk formulas used by SBFL as reported in [2], as the suspiciousness formulas. As studied in [2], different risk formulas of SBFL have different performance in predicting a faulty statement. We wish to further investigate the performance of such risk formulas of SBFL when used as the suspiciousness formulas in predicting which test case is a failed test case. We anticipate to report interesting results to the research community in future.

ACKNOWLEDGMENT

The project is supported by Natural Science Foundation of Jiangsu Province, China (No:BK20160769, BK20141072) and China Postdoctoral Science Foundation (No: 45055).

REFERENCES

[1] X. Xie, W.E. Wong, T.Y. Chen, and B. Xu, "Spectrum-based fault localization: testing oracles are no longer mandatory", *Proc. of Eleventh International Conf. on Quality Software*, pp. 1–10, 2011.

[2] F.T. Chan, T.Y. Chen, S.C. Cheung, M.F. Lau, and S.M. Yiu, "Application of metamorphic testing in numerical analysis", *Proc. Of IASTED International Conf. on Soft. Eng.*, pp. 191–197, 1998.

[3] S. Segura, G. Fraser, A.B. Sanchez, and A. Ruiz-Cortés, "A survey on metamorphic testing," *IEEE Trans. on Softw. Eng.*, vol. 42, p. 805–824, 2016.

[4] X. Xie, J. Ho, C. Murphy, G. Kaiser, B. Xu, and T.Y. Chen, "Application of metamorphic testing to supervised classifiers", *Proc. of Ninth International Conf. on Quality Software*, pp. 135–144, 2009.

[5] Y. Jia and M. Harman, "A customizable runtime-optimized higher order mutation testing tool for the full C language", *Proceedings of the Third TAIC PART 08*, Windsor, UK, 2008.

[6] Y.-S. Ma, J. Offutt, and Y.-R. Kwon, "MuJava: an automated class mutation system", *Software Testing, Verification and Reliability*, vol. 15, p. 97–133, 2005.