SCIENCE CHINA
Information Sciences

• INSIGHT •
Special Focus on Advanced Technology of Software
May 2016, Vol. 59 050108:1–050108:2
doi: 10.1007/s11432-016-5544-6

Metamorphic testing as a test case selection strategy

Dave TOWEY[1]*, Yunwei DONG[2], Chang-Ai SUN[3] & Tsong Yueh CHEN[4]

[1]*School of Computer Science, The University of Nottingham Ningbo China, Ningbo* 315100*, China;*
[2]*School of Computer Science and Engineering, Northwestern Polytechnical University, Xi'an* 710072*, China;*
[3]*School of Computer and Communication Engineering, University of Science and Technology Beijing, Beijing* 100083*, China;*
[4]*Department of Computer Science and Software Engineering, Swinburne University of Technology, Victoria* 3122*, Australia*

Received December 29, 2015; accepted January 18, 2016; published online April 8, 2016

Citation Towey D, Dong Y W, Sun C-A, et al. Metamorphic testing as a test case selection strategy. Sci China Inf Sci, 2016, 59(5): 050108, doi: 10.1007/s11432-016-5544-6

Software testing is a well-known and popular approach to ensuring quality software. It involves execution of the software under test (SUT), in an attempt to either offer some degree of confidence that the software is relatively bug free, or to actually uncover bugs or problems (called failures). Software testing has been studied for many years, and has generated a large body of approaches, methods, and techniques [1]. An assumption underlying many of these approaches, however, is that it is possible (and practical) to identify the correctness of the SUT, when tested. Unfortunately, this assumption does not always hold true, and software testers often face the challenge of testing software without being able to say whether or not the output is correct — a situation known as the Oracle Problem.

An innovative approach to dealing with the Oracle Problem is metamorphic testing (MT) [2], which, instead of attempting to verify the correctness of individual outputs of the SUT, uses identified relationships of the SUT's problem domain. These relationships, called metamorphic relations (MRs), are often known properties of the functionality to be implemented by the software. If testing an implementation of the sine function, then an example of an MR is the metamorphic property of $\sin(x) = \sin(2\pi + x)$. Although exact determination of the output's correctness for any arbitrary value of x may not be possible, a comparison of this output with that for $(2\pi + x)$ should be relatively easy. With such an MR, if the outputs are not equal (after consideration of possible rounding errors), then a bug must be present. MT has been proven to effectively alleviate the Oracle Problem in many different domains [3–6].

In addition to alleviating the Oracle Problem, though, MT can be applied as a test case selection methodology in its own right. Even in situations where the SUT's output can be verified — that is, an oracle exists, and can be used — MT has demonstrated excellent fault detection abilities. To illustrate this, some recent cases will next be presented.

The Siemens test suite[1)] is a collection of programs often used by the software testing community to study the effectiveness of various testing approaches, and therefore represents some very well-tested software. Furthermore, as new approaches are proposed, they are often applied to this suite, meaning that a great deal of popular and effective test case selection strategies have been used to test

* Corresponding author (email: Dave.Towey@nottingham.edu.cn)
The authors declare that they have no conflict of interest.

1) SIR. Software-artifact infrastructure repository. http://sir.unl.edu/portal/index.php.

these programs. However, in two recent, independent studies, MT was able to detect previously unknown bugs in three of the Siemens' programs [7]. We believe that it is due to the fact that MT generates test cases from a different perspective, one which was not considered by the other test case selection strategies, that enabled detection of these new failures [7]. This bug detection — of previously unknown bugs in such well-tested software — is very strong support for the effectiveness of MT.

A second example is "based on a particularly clever application of metamorphic testing" [8]. Compilers are used to generate object code, and it is therefore very important that they are bug-free. Le et al. [9] effectively used an MR as follows: For a given compiler C, let O be the object program generated when C is used to compile source program P. Execute O on a particular input x, and identify the code in P which was not executed. Construct another source program P' from P by pruning some of the non-executed codes. Then, use C to compile P', generating another object program O'. Execute O' on input x. If the compiler is implemented correctly, then the outputs for executing x with object programs O and O' should be the same; otherwise the compiler is buggy. At the time of publication, the authors had generated "147 confirmed, unique bug reports for GCC and LLVM alone", of which already more than one hundred had been fixed [9] — interested readers may refer to[2)] for updates on the numbers of identified and fixed bugs. The success of MT in revealing such a huge amount of bugs for these two popular compilers is a clear evidence of the effectiveness of MT when used as a test case selection strategy.

A third example of MT's ability to detect bugs was when MT was used to analyze feature models—representations of (valid) features of a software product. When Segura et al. [10] applied MT to the analysis of feature models, they found real-life defects in actual feature model analysis tools: Two bugs in FaMa; and two in SPLOT.

A final example of MT's ability to detect failure comes from NASA: The NASA Data Access Toolkit allows analysts to query a large database of telemetry data [11]. MT was applied to generate equivalent queries, which were executed in different ways. Because the queries were equivalent, different outputs meant the presence of a bug. According to Lindvall et al. [11], this MT approach detected several "confirmed issues", and is a "promising approach".

As has been clearly shown, MT is not only an excellent approach for alleviating the Oracle Problem, but is also an effective test case selection strategy in its own right. Its future application to the testing of all types of software, not only those lacking oracles, promises to be an exciting research area.

References

1 Anand S, Burke E, Chen T Y, et al. An orchestrated survey on automated software test case generation. J Syst Soft, 2013, 86: 1978–2001

2 Chen T Y, Cheung S C, Yiu S M. Metamorphic Testing: a New Approach for Generating Next Test Cases. Technical Report HKUST-CS98-01. 1998

3 Chen T Y, Ho J W K, Liu H, et al. An innovative approach for testing bioinformatics programs using metamorphic testing. BMC Bioinformatics, 2009, 10: 1–12

4 Kuo F C, Chen T Y, Tam W K. Testing embedded software by metamorphic testing: a wireless metering system case study. In: Proceedings of the 36th IEEE Conference on Local Computer Networks (LCN), Bonn, 2011. 295–298

5 Sun C, Wang G, Mu B, et al. A metamorphic relation-based approach to testing web services without oracles. Int J Web Serv Res, 2012, 9: 51–73

6 Pullum L, Ozmen O. Early results from metamorphic testing of epidemiological models. In: Proceedings of ASE/IEEE International Conference on BioMedical Computing (BioMedCom), Washington, 2012. 62–67

7 Chen T Y, Kuo F C, Towey D, et al. A revisit of three studies related to random testing. Sci China Inf Sci, 2015, 58: 052104

8 Regehr J. Finding compiler bugs by removing dead code. http://blog.regehr.org/archives/1161. 2014

9 Le V, Afshari M, Su Z. Compiler validation via equivalence modulo inputs. In: Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'14). New York: ACM, 2014. 216–226

10 Segura S, Hierons R M, Benavides D, et al. Automated metamorphic testing on the analyses of feature models. Inf Softw Tech, 2011, 53: 245–258

11 Lindvall M, Ganesan D, Árdal R, et al. Metamorphic model-based testing applied on NASA DAT: an experience report. In: Proceedings of the 37th International Conference on Software Engineering (ICSE'15), Florence, 2015. 129–138

2) Su Z. Zhendong Su's homepage. http://web.cs.ucdavis.edu/~su/.