HKU CS Tech Report TR-2021-02

Postprint of paper in Ideas, Visions and Reflections Track, *Proceedings of the ACM Joint European Software Engineering Conference and Symposium on Foundations of Software Engineering* (*ESEC/FSE '21*), ACM, New York, NY, pp. 4087–4090 (2021)

New Visions on Metamorphic Testing after a Quarter of a Century of Inception

Tsong Yueh Chen
Department of Computer Science & Software Engineering
Swinburne University of Technology, Australia
tychen@swin.edu.au

T. H. Tse
Department of Computer Science
The University of Hong Kong, Hong Kong
thtse@cs.hku.hk

ABSTRACT

Metamorphic testing (MT) was introduced about a quarter of a century ago. It is increasingly being accepted by researchers and the industry as a useful testing technique. The studies, research results, applications, and extensions of MT have given us many insights and visions for its future. Our visions include: MRs will be a practical means to top up test case generation techniques, beyond the alleviation of the test oracle problem; MT will not only be a standalone technique, but conveniently integrated with other methods; MT and MRs will evolve beyond software testing, or even beyond verification; MRs may be anything that you can imagine, beyond the necessary properties of algorithms; MT research will be beyond empirical studies and move toward a theoretical foundation; MT will not only bring new concepts to software testing but also new concepts to other disciplines; MRs will alleviate the reliable test set problem beyond traditional approaches. These visions may help researchers explore the challenges and opportunities for MT in the next decade.

CCS CONCEPTS

• **Software and its engineering → Software testing and debugging**.

KEYWORDS

Metamorphic testing, Metamorphic relation, Testing, Test oracle, Reliable test set, Proving, Debugging

ACM Reference Format:
Tsong Yueh Chen and T. H. Tse. 2021. New Visions on Metamorphic Testing after a Quarter of a Century of Inception. In *Proceedings of the 29th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '21), August 23–28, 2021, Athens, Greece.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3468264.3473136

1 INTRODUCTION

Metamorphic testing (MT) [3] was introduced by Tsong Yueh Chen about a quarter of a century ago, as motivated by the question "Are successful test cases really useless?" Following up on the fundamental concept, more than 440 papers have been published by various researchers and presented fruitful results. For instance, Le et al. [16] detected more than 100 faults in existing GCC and LLVM compilers by means of the innovative "equivalence modulo inputs" technique, which is effectively MT. Another interesting example is, through the use of "metamorphic robustness testing", Zhou et al. [35] revealed that the presence of hyphens in paper titles inadvertently reduces citation counts and journal impact factors. This "bizarre" finding resulted in immense coverage in the international electronic media as well as strong rebuttal by Web of Science [7]. Yet another (sad) example is that Zhou and Sun [33], using MT, "detected fatal software faults in the LiDAR obstacle-perception module of self-driving cars and reported the alarming results eight days before Uber's deadly crash in Tempe, AZ, in March 2018."

Not only is MT recognized by researchers, but is also increasingly being accepted by the industry. Some notable examples include Google's high-profile acquisition [10] of GraphicsFuzz [8] (which applies MT and fuzzing to test graphics drivers in mobile phones); Accenture's patent on machine learning verification [9] (which significantly reduces the number of test cases to identify bugs); the testing of web enabled simulation at scale in Facebook [1]; the data analytic system in Adobe Inc. [14]; the data access toolkit in NASA [18]; the storm water management model system in the US Environmental Protection Agency [17]; and the epidemiological model simulation system in Oak Ridge National Laboratory sponsored by the US Department of Energy [21].

Two major surveys [3, 22] have been published to review the progress of MT. We will adopt a different approach in this ESEC/FSE-IVR paper: We will present our insights and visions on MT in many aspects unforeseen by the originator a quarter of a century ago. We hope that our analysis will help researchers focus their studies on the most appropriate direction.

2 METAMORPHIC TESTING AND METAMORPHIC RELATIONS

In this section, we will set the scene by briefly highlighting the concepts of metamorphic testing, metamorphic relations, and metamorphic groups of inputs. Readers may refer to Chen et al. [3] and Segura et al. [22] for more details.

Metamorphic testing (MT) was proposed with a view to extracting useful information from successful test cases that do not reveal failures. It involves multiple executions of two or more test cases. For ease of understanding, we will only discuss the execution of two test cases in this paper. The input and output of the first test case are called the source input and output, while those of the other test case are called the *follow-up* input and output. These test cases are

ESEC/FSE '21, August 23–28, 2021, Athens, Greece

ACM ISBN 978-1-4503-8562-6/21/08…$15.00
https://doi.org/10.1145/3468264.3473136

linked up by *metamorphic relations* (MRs), which are determined by some necessary properties of the algorithm of the program under test. The pair of source and follow-up inputs is called a *metamorphic group* (MG) of inputs.

In software testing, a *test oracle* is the mechanism to determine whether the outputs from a program is correct. A program is said to be non-testable if (a) an oracle does not exist, or (b) it takes too much time to apply. We refer to such situations as the *oracle problem.*

Consider, for instance, a program that implements the sine function. It accepts an angle x in degrees to 99 places after decimal and returns $\sin(x)$ with the same number of decimal places. There is no simple oracle to determine whether the output is erroneous. Taking note of MT guidelines, we treat x and $\sin(x)$ as source input and output, and apply the basic trigonometric property that $\sin(x) = \sin(180 - x)$ for any angle x in degrees. It can be expressed as an MR that relates the source and follow-up test cases:

$$\text{If } x' = 180 - x, \text{ then } \sin(x') = \sin(x).$$

Testing the program again using a follow-up input $x' = 180 - x$, if $\sin(x') \neq \sin(x)$, it will indicate that the MR has been violated, thus revealing a failure.

In addition to alleviating the oracle problem, MRs are also useful in generating test cases based on successful test cases.

3 NEW VISIONS ON METAMORPHIC TESTING

3.1 Beyond Alleviation of Test Oracle Problem: A Top-up Test Case Generation Method

MT started off as a test case generation technique, but shortly became better known as a method to address the test oracle problem. We envisage that MT's role as a good test case generation technique will continue to grow and become as popular as its function to alleviate the test oracle problem.

The original motivation behind MT is how to explore the useful information associated with a test case that does not reveal any failure. Therefore, MT was first designed as a test case generation method. Shortly after its invention, it became obvious that MT could alleviate the oracle problem. Since then, the focus on MT has been shifted to addressing this problem, that is, the use of MT to test non-testable software.

MT appears to be the only approach that can address the two key problems in software testing, namely the oracle problem and the reliable test set problem (see Section 3.7). The original intuition for MT is: The source test cases are generated by a specific test case generation method, referred to as the original test case generation method. Then, follow-up test cases are generated by a method built on the given MR and the original test case generation method. In other words, given any test case generation method, MT provides a generic approach to transforming it into an "enhanced" version, which is expected to have a better failure detection capability than the original method. Recent studies [2, 25, 32] have reported that follow-up test cases are more effective in revealing failures than their corresponding source test cases. We envisage that more experimental and theoretical studies will be conducted to further investigate the failure detection capabilities between the source and follow-up test cases. More attention will be focused on the initial aim behind the proposal of MT, that is, how to use MT as an effective test case generation method, regardless whether the software is testable or non-testable. In the future, if one has decided to adopt a particular test case generation technique, it will be worthwhile to consider topping it up with appropriate MRs.

3.2 Beyond a Standalone Technique: Integration with Other Methods

Originally developed as a standalone technique, MT has successfully integrated with other methods. We envisage that MT will be a popular integration partner with many more methods.

Even though MT was developed as a standalone technique, it can easily be integrated with other techniques, mainly because its concept is simple and its process is straightforward. The only arguable assumption in MT is the existence and availability of MRs, which are defined as necessary properties involving multiple inputs and the corresponding outputs of the algorithm for the program under test. We cannot conceive of any application that does not have desirable properties that are necessary. As a reminder, MRs are only *necessary* properties, rather than *necessary and sufficient* properties. Therefore, there are normally many potential MRs, and it should not be too difficult to identify at least some of them. Moreover, the MT process only involves program executions and checking the relations among multiple inputs and their computed outputs. As there is no need for any preconditions except specific MRs, and because of the simplicity of the process, MT can easily be integrated with other methods [15]. Existing work has covered the integration of MT with debugging [5], fault localization [28], program repair [15], and symbolic execution [5]. We envisage that it will become a standard research question for researchers to consider whether, through the use of MRs, the applicability of their developed methods can be extended to programs without test oracles, even if the original methods assume the presence of oracles. This may as well become a common question to be raised by reviewers for papers assuming the need for oracles.

As described in [3], the integration can be facilitated through two steps, namely, (a) the correspondence between a single test case and an MG of test cases, and (b) the correspondence between the pass/fail outcome of the single test case and the satisfaction/violation outcome of the MG with respect to an MR. In conventional debugging with slicing, for instance, we need only debug the slices related to failure-revealing inputs (instead of the whole program), because we know that the fault must reside within these slices. When we integrate MT and debugging with slicing, we need only debug the slices related to violation-revealing MGs. In short, the focus is shifted from failure-revealing inputs to violation-revealing MGs.

3.3 Beyond Testing: Also Proving and Debugging

Originally developed as a software testing technique, MT has migrated into proving and debugging as additional means of verification. It appears to be the only technique applicable to all three areas of verification, namely testing, proving, and debugging.

MT was first proposed as a software testing technique for verification, by generating follow-up test cases and checking against the MRs. Like the limitation of traditional software testing, it can only reveal the failures of a program with respect to MRs, but cannot prove the correctness of the program. In other words, MT can only show that an MR does not hold for the program for some MGs, but cannot show that MRs always hold for the program for any eligible MGs. It is therefore natural to go beyond testing MRs to the proving of MRs. In this way, MT evolves into a proving technique [5]. More importantly, when MRs are proven not to hold for a program, constraints on inputs that lead to the invalidity of the MRs can be obtained. Such constraints provide useful information for program debugging and repair [5].

During traditional testing, we may detect an erroneous result for a specific input of, say, $x = 5$. Debuggers will then attempt to localize the fault and repair the program based on this specific input. Suppose we apply MT instead. We would reveal the violation of an MR when, say, x is greater than 3 but smaller than 6. Obviously, the condition "$3 < x < 6$" is more informative than the specific input "$x = 5$" in debugging and repair.

The use of MT as proving and debugging techniques will grow because it provides a very different perspective when compared with traditional proving and debugging, which are driven by specific inputs; whereas MRs involve conditions for multiple inputs. We envisage that MT will show its popularity in all aspects of verification: not only testing, but also proving and debugging. To the best of our knowledge, MT is the only technique that is applicable to all three areas of verification.

3.4 Beyond Necessary Properties of the Algorithms: Anything You Can Imagine

Although MRs were first defined as the necessary properties of the algorithm whose implementation will be tested, the definition, roles, and uses have evolved beyond verification. We envisage that new roles and uses of MRs will continue to emerge.

MRs were originally defined as the necessary properties of program algorithms. In the context of validation [27], MRs can be defined for the expectations from the users' perspective. In the context of system assessment or selection [29], MRs can be defined for the evaluation, adequacy, or appropriateness criteria. In the context of software understanding [34], MRs can help user comprehension in the absence of a thorough specification. Thus, diverse definitions, roles, and uses of MRs have emerged as the MT concept is being extended to various areas. We envisage a continued growth of the diversified roles and usages of MRs to capture various kinds of information through the powerful mathematical "relations". They do not have any limitation except perhaps our own imagination.

3.5 Beyond Empirical Research: Toward a Theoretical Foundation for MT

Because of its simplicity, there has been a long misconception that MT is unlikely to have an enriched and sound theoretical foundation with formally predictable consequences. Nevertheless, researchers have found that the development of a solid foundation theory for MT is feasible and useful. For instance, a recent theoretical analysis unveils sufficient conditions for *composite* MRs to be more cost effective than *component* MRs [20] (in terms of the ratio of the number of faults detected to the number of program executions). This result is important because it identifies the precise situations where a composite MR will preserve the fault detection capability of individual component MRs. This saves test resources without the need to compromise the fault detection capability. Another key area is a foundation for the automatic construction of MRs, which has received growing attention as more and more MR construction methods are being developed [23, 30]. Yet another area is the concept of diversity for MRs [19], which plays a major role in their effectiveness. It is well known that, as necessary properties, MRs can be expressed as a partially ordered set. Hence, for any two given MRs that have a specific connection in this partial ordering, what is the relationship between their fault detection effectiveness? There are so many interesting (although challenging) questions to be investigated with a view to building a theoretical foundation for MT, which will in turn lead to various impacts on software quality.

3.6 Beyond New Concepts in Software Testing: New Concepts in Other Disciplines As Well

Not only has MT introduced new ideas in software testing, but also new concepts in other disciplines. We envisage more new concepts will continue to develop in other disciplines along with the evolution of MT and MR.

The fundamentals for MT have gradually been developed. The concepts of diversity of MRs, composition of MRs, adequacy of MRs, fault detection effectiveness of MT, and the likelihood that an input in a violating MG is a failing input, have gradually been established to enrich the solid foundation. At the same time, the emergence of new concepts is not just restricted to software testing. New concepts are being brought by MT in other disciplines as well. For instance, slicing [24, 26] is a very important concept in program analysis. A new type of slice [28] was proposed for MT as a counterpart of traditional execution slices. Similarly, MT-based counterparts of other kinds of traditional slices are envisaged, resulting in a new family of metamorphic slices in parallel with the original family of traditional slices. This will definitely enrich the notion of slicing and enhance its applicability.

3.7 Beyond the Traditional Approach to Alleviating the Reliable Test Set Problem: Selecting MRs Rather Than Test Cases

It is well known that testing can only reveal the presence of faults in a piece of software, but cannot guarantee its correctness. More specifically, Howden [13] has proven that it is impossible to find any algorithm such that, for any given program P, it can construct a reliable test set T whose successful execution will imply the correctness of P, unless T is the entire input domain. This is referred to as the reliable test set problem.

Even though the problem cannot be completely solved, numerous test case selection or generation techniques have been developed for the last several decades to *improve* test adequacy and effectiveness. Nevertheless, research on test case selection appears to have reached a slow moving stage, as no innovative technique has been developed for quite a while. Research has shown that a small number of *diverse* MRs is almost as effective as the whole set of MRs [19] in ensuring software quality. Hence, we envisage that MR selection (instead of test case selection) will emerge as a fundamental task for effective and efficient software testing.

4 CONCLUDING REMARKS

Along with the evolution of MT, we anticipate that more and more interesting, significant, and influential research will continue to emerge, as evidenced by the continuing growth of papers on MT. New concepts will be developed to enrich the theoretical foundation for MT as well as other disciplines. We envisage to have better knowledge and understanding on how to define and generate MRs, how to composite MRs, and how to select a group of diverse MRs for cost-effective testing. The selection and generation of MRs will become as important and as fundamental as the selection and generation of test cases. Furthermore, the following will become standard questions for researchers to answer:

(1) How to use MRs to top up your test case generation technique?
(2) How to integrate MT with your technique that assumes a test oracle, so that the enhanced method will be applicable to non-testable systems?

We envision that MT may become a cutting-edge approach to the quality assurance of big data analytics [6], deep and machine learning systems [4, 11, 31], and complex simulations [12, 17], which are some of the major types of systems in our daily life for the next few decades, as already evidenced by some of its initial successes in these areas.

ACKNOWLEDGMENTS

T. Y. Chen would like to thank David Rosenblum for his encouragement and confidence in metamorphic testing since the early days of inception. This project is supported by a grant from Facebook and a grant from Australian Research Council (DP210102447).

REFERENCES

[1] J. Ahlgren, M. Berezin, et al. 2021. Testing web enabled simulation at scale using metamorphic testing. *Proceedings of the 2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP'21)* (2021), 140–149.
[2] A. Chan, L. Ma, F. Juefei-Xu, Y. S. Ong, X. Xie, M. Xue, and Y. Liu. 2021. Breaking neural reasoning architectures with metamorphic relation-based adversarial examples. *IEEE Trans. Neural Net. Learn. Sys.* (2021). https://doi.org/doi:10.1109/TNNLS.2021.3072166
[3] T. Y. Chen, F. C. Kuo, H. Liu, P. L. Poon, D. Towey, T. H. Tse, and Z. Q. Zhou. 2018. Metamorphic testing: A review of challenges and opportunities. *ACM Comp. Surv.* 51(1) (2018), 4:1–4:27.
[4] T. Y. Chen, P. L. Poon, K. Qiu, Z. Zheng, and J. Zhou. 2021. Use of metamorphic relations as knowledge carriers to train deep neural networks. *arXiv preprint* (2021), arXiv:2104.04718.
[5] T. Y. Chen, T. H. Tse, and Z. Q. Zhou. 2011. Semi-proving: An integrated method for program proving, testing, and debugging. *IEEE Trans. Soft. Eng.* 37(1) (2011), 109–125.
[6] A. Davoudian and M. Liu. 2020. Big data systems: A software engineering perspective. *ACM Comp. Surv.* 53(5) (2020), 110:1–110:39.
[7] P. Dockrill. 2019. Study claims one punctuation mark has been skewing our scientific ranking system. ScienceAlert. *Retrieved from https://www.sciencealert.com/hyphens-break-our-entire-system-of-scientific-ranking-new-analysis-reveals* (2019).
[8] A. F. Donaldson, H. Evrard, A. Lascu, and P. Thomson. 2017. Automated testing of graphics shader compilers. *Proc. ACM Program. Lang. 1(OOPSLA)* (2017), 93:1–93:29.
[9] A. Dwarakanath, M. Ahuja, S. Sikand, R. M. Rao, R. P. J. C. Bose, N. Dubash, and S. Podder. 2018. Identifying implementation bugs in machine learning based image classifiers using metamorphic testing. *Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA'18)* (2018), 118–128.
[10] Google-GraphicsFuzz. 2018. GraphicsFuzz is acquired by Google. *Retrieved from https://www.graphicsfuzz.com/* (2018).
[11] P. He, C. Meister, and Z. Su. 2020. Structure-invariant testing for machine translation. *Proceedings of the 42nd International Conference on Software Engineering (ICSE'20)* (2020), 961–973.
[12] X. He, X. Wang, J. Shi, and Y. Liu. 2020. Testing high performance numerical simulation programs: Experience, lessons learned, and open issues. *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA'20)* (2020), 502–515.
[13] W. E. Howden. 1976. Reliability of the path analysis testing strategy. *IEEE Trans. Soft. Eng.* 2(3) (1976), 208–215.
[14] D. C. Jarman, Z. Q. Zhou, and T. Y. Chen. 2017. Metamorphic testing for Adobe data analytics software. *Proceedings of the 2nd International Workshop on Metamorphic Testing (MET'17)* (2017), 21–27.
[15] M. Jiang, T. Y. Chen, F. C. Kuo, D. Towey, and Z. Ding. 2017. A metamorphic testing approach for supporting program repair without the need for a test oracle. *J. Soft. Sys.* 126 (2017), 127–140.
[16] V. Le, M. Afshari, and Z. Su. 2014. Compiler validation via equivalence modulo inputs. *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'14)* (2014), 216–226.
[17] X. Lin, M. Simon, and N. Niu. 2021. Scientific software testing goes serverless: Creating and invoking metamorphic functions. *IEEE Soft.* 38(1) (2021), 61–67.
[18] M. Lindvall, D. Ganesan, R. Ardal, and R. E. Wiegand. 2015. Metamorphic model-based testing applied on NASA DAT: An experience report. *Proceedings of the 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering (ICSE'15)* 2 (2015), 129–138.
[19] H. Liu, F. C. Kuo, D. Towey, and T. Y. Chen. 2014. How effectively does metamorphic testing alleviate the oracle problem? *IEEE Trans. Soft. Eng.* 40(1) (2014), 4–22.
[20] K. Qiu, Z. Zheng, T. Y. Chen, and P. L. Poon. 2020. Theoretical and empirical analyses of the effectiveness of metamorphic relation composition. *IEEE Trans. Soft. Eng.* (2020). https://doi.org/doi:10.1109/TSE.2020.3009698
[21] A. Ramanathan, C. A. Steed, and L. L. Pullum. 2012. Verification of compartmental epidemiological models using metamorphic testing, model checking and visual analytics. *Proceedings of the 2012 ASE/IEEE International Conference on BioMedical Computing (BioMedCom'12)* (2012), 68–73.
[22] S. Segura, G. Fraser, A. B. Sanchez, and A. Ruiz-Cortes. 2016. A survey on metamorphic testing. *IEEE Trans. Soft. Eng.* 42(9) (2016), 805–824.
[23] S. Segura, J. A. Parejo, J. Troya, and A. Ruiz-Cortes. 2018. Metamorphic testing of RESTful web APIs. *IEEE Trans. Soft. Eng.* 44(11) (2018), 1083–1099.
[24] J. Silva. 2012. A vocabulary of program slicing-based techniques. *ACM Comp. Surv.* 44(3) (2012), 12:1–12:41.
[25] Y. Tian, K. Pei, S. Jana, and B. Ray. 2018. DeepTest: Automated testing of deep-neural-network-driven autonomous cars. *Proceedings of the 40th International Conference on Software Engineering (ICSE'18)* (2018), 303–314.
[26] M. Weiser. 1984. Program slicing. *IEEE Trans. Soft. Eng.* 10(4) (1984), 352–357.
[27] X. Xie, J. W. K. Ho, C. Murphy, G. Kaiser, B. Xu, and T. Y. Chen. 2011. Testing and validating machine learning classifiers by metamorphic testing. *J. Sys. Soft.* 84(4) (2011), 544–558.
[28] X. Xie, W. E. Wong, T. Y. Chen, and B. Xu. 2013. Metamorphic slice: An application in spectrum-based fault localization. *Info. Soft. Tech.* 55(5) (2013), 866–879.
[29] X. Xie, Z. Zhang, T. Y. Chen, Y. Liu, P. L. Poon, and B. Xu. 2020. METTLE: A metamorphic testing approach to assessing and validating unsupervised machine learning systems. *IEEE Trans. Reliab.* 69(4) (2020), 1293–1322.
[30] B. Zhang, H. Zhang, J. Chen, D. Hao, and P. Moscato. 2019. Automatic discovery and cleansing of numerical metamorphic relations. *Proceedings of 2019 IEEE International Conference on Software Maintenance and Evolution (ICSME'19)* (2019), 235–245.
[31] J. M. Zhang, M. Harman, L. Ma, and Y. Liu. 2020. Machine learning testing: Survey, landscapes and horizons. *IEEE Trans. Soft. Eng.* (2020), doi:10.1109/TSE.2019.2962027.
[32] M. Zhang, Y. Zhang, L. Zhang, C. Liu, and S. Khurshid. 2018. DeepRoad: GAN-based metamorphic testing and input validation framework for autonomous driving systems. *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering (ASE'18)* (2018), 132–142.
[33] Z. Q. Zhou and L. Sun. 2019. Metamorphic testing of driverless cars. *Comms. ACM* 62(3) (2019), 61–67.
[34] Z. Q. Zhou, L. Sun, T. Y. Chen, and D. Towey. 2020. Metamorphic relations for enhancing system understanding and use. *IEEE Trans. Soft. Eng.* 46(10) (2020), 1120–1154.
[35] Z. Q. Zhou, T. H. Tse, and M. Witheridge. 2021. Metamorphic robustness testing: Exposing hidden defects in citation statistics and journal impact factors. *IEEE Trans. Soft. Eng.* 47(6) (2021), 1164–1183.