This is a preprint. Please cite as:
Zhihao Ying, Dave Towey, Anthony Bellotti, Zhi Quan Zhou and Tsong Yueh Chen (2022), "Using Metamorphic Relation Violation Regions to Support a Simulation Framework for the Process of Metamorphic Testing", in Proceedings of the IEEE 46th Annual Computers, Software, and Applications Conference, COMPSAC 2022, pp. 1722-1727.

Using Metamorphic Relation Violation Regions to Support a Simulation Framework for the Process of Metamorphic Testing

Zhihao Ying, Anthony Bellotti and Dave Towey*
School of Computer Science
University of Nottingham Ningbo China
Zhejiang 315100
People's Republic of China
{scyym1, Anthony-Graham.Bellotti, Dave.Towey}@nottingham.edu.cn

Tsong Yueh Chen
Department of Computer Science and Software Engineering
Swinburne University of Technology
Hawthorn, VIC 3122
Australia
tychen@swin.edu.au

Zhi Quan Zhou
School of Computing and Information Technology
University of Wollongong
Wollongong, NSW 2522
Australia
zhiquan@uow.edu.au

***Abstract*—Metamorphic testing (MT) has been growing in popularity, but it can still be quite challenging and time-consuming to assess its performance. Typical approaches to performance assessment can require a series of steps, and depend on a variety of factors, often requiring serendipity. This can be a bottleneck for some aspects of MT research. Central to MT, metamorphic relations (MRs) represent necessary properties of the system under test (SUT). In traditional software testing, simulations are often employed to examine and compare the performance of different testing strategies. However, these simulations are typically designed based on the assumed availability (and applicability) of a test oracle — a mechanism to decide the correctness of the SUT output or behaviour. A key reason for the popularity of MT is its proven record of effective software testing, *without* the need for a test oracle. This strength, however, also means that traditional ways of using simulations to analyse software testing approaches are not applicable for MT. This lack of cheap and fast ways to conduct simulation analyses of MT is a hurdle for many aspects of MT research, and may be an obstacle to its more widespread adoption. To address this, in this paper we introduce the concept of MR-violation regions (MRVRs), and show how they can be used for a certain category of MRs, Deterministic MRs (DMRs), to build simulation tools for MT. We analyse the differences between MRVRs and traditional, oracle-defined failure regions; and report on a preliminary case study exploring MRVRs in numerical-input-domain systems from previous MT studies. We anticipate that the proposed MT simulation framework may facilitate more research into MT, and may help lead to its more widespread adoption.**

***Index Terms*—software quality assurance, oracle problem, metamorphic relations (MRs), metamorphic relation violation region (MRVR), metamorphic testing, simulations**

I. Introduction

Computer software is becoming more complex and sophisticated, but, in many cases, the related software quality assurance processes are not keeping pace, resulting in an increasing number of challenges. One of the fundamental challenges is the oracle problem [1]–[4]. In traditional software testing, the oracle is a mechanism for checking the correctness of the output/behaviour of the system under test (SUT). If the oracle is not available, or is too expensive to be used, then this SUT is said to face the oracle problem [1], [4]. Traditional software testing methods are not applicable for testing systems facing the oracle problem, but metamorphic testing (MT) is a method that has a record of being able to alleviate it [2], [3], [5]. MT detects software failures by checking the metamorphic relations (MRs) among multiple test cases, where MRs represent the necessary properties of the SUT [2], [3], [6]. Given an MR, the relevant source test cases (STCs) and follow-up test cases (FTCs) are referred to as metamorphic groups (MGs) [3]. According to the number of STCs and FTCs, the MRs generally can be divided into the following four classes [3], [7]: (1) 1-1 MRs, where only one STC is used to generate only one FTC; (2) 1-N MRs, where only one STC is used to generate N FTCs ($N > 1$); (3) M-1 MRs, where M STCs ($M > 1$) are used to generate only one FTC; and (4) M-N MRs, where M STCs ($M > 1$) are used to generate N FTCs ($N > 1$).

The successful implementation of MT relies heavily on the MRs and MGs used [2], [7], [8]. Furthermore, evaluation of the MT performance can involve a number of steps, such as, for example, the following:

1) Identify an SUT and create faulty copies (called mutants) by inserting artificial faults into the system, manually or with an automatic mutation tool [9].
2) Identify MRs for the SUT.
3) Generate the MGs (the STCs and FTCs).
4) Execute the generated test cases against the mutants.
5) Examine the outputs with reference to the given MRs, checking for violations.

Obviously, this can be a difficult and time-consuming process.

In traditional software testing, the approach of simulation is commonly used to examine the performance of the testing

*Dave Towey is the corresponding author for this paper.

method [10], and can reduce much of the difficulty and time overheads. The simulation approach typically uses artificial failure regions, and can evaluate the performance by checking whether or not the generated test cases are located inside these failure regions [10]. In normal testing, failure regions are those parts of the input domain that are failure-revealing inputs: Their execution causes the SUT to produce unexpected behaviour or output — and thus, the test case fails. In simulations, SUT execution is not necessary, the location of the test case is sufficient to determine whether or not it fails.

Because MT does not assume the existence of an oracle [10], and checks for MR violations (not for test-case failures), traditional approaches for using simulations are not readily applicable. This lack of easy access to simulations can hinder the fast development and exploration of some aspects of MT research. This paper addresses this challenge by introducing the concept of MR-violating regions (MRVRs), and using them to support simulation analysis and testing for a particular category of MRs. The simulation approach can reduce the time required to evaluate the MT performance, and make it easier for testers/researchers to set up MT-related experiments. As reported in previous studies [7], [8], [11], the quality of MGs is likely to impact the performance of MT. However, evaluation of the MT test efficiency (the time required for generation and execution of each test case [12]) and effectiveness (fault-detection capability [12], [13]) is often a difficult and time-consuming task [2]. Being able to simulate the MT process should enable testers/researchers to more easily and quickly achieve these aims, supporting faster and deeper investigation into MT.

This paper also briefly explores the relationship between failure regions (from traditional, oracle-based software testing) and MRVRs, and proposes three categories of MRVRs: block; point; and strip. We also report on a preliminary investigation into the presence of these MRVR types in systems with numerical inputs, from previous MT-related studies.

The rest of this paper is structured as follows: Section II introduces the background information for traditional software testing and simulations. Section III introduces our proposed simulation framework for MT. As part of our framework presentation, we also introduce the concepts of Deterministic MRs (Section III-A), and MR-violation regions (MRVRs) (Section III-B). In Section III-C, we compare and contrast MRVRs with traditional failure regions, explaining how they are not expected to be the same. Section IV reports on a case study exploring the potential MRVRs in systems with numerical inputs. Finally, Section V offers a conclusion, and discusses limitations and future work.

II. Background

In software testing, when a program developer makes a mistake, a fault in this program may be produced, and a failure may be detected if this fault is encountered [14]. In other words, after the execution of a test case tc, a failure is revealed/detected if the output or behaviour of the SUT is different from expected. In this situation, tc is called a *failure-causing test case*. The set of all failure-causing test cases for a given SUT, as a proportion of all possible test cases in the input domain, is called the *failure rate* [3], [10]. *Random testing* (RT) is a popular black-box software testing technique, producing random, independent test cases for the SUT [15], [16]. It has been reported that failure-causing test cases tend to cluster into contiguous regions called failure regions [17]–[22]. The neighbours of a failure-causing test case should have a high probability of also being failure-causing; and the neighbours of a non-failure-causing test case should have a high probability of being non-failure-causing. Therefore, if new test cases are selected from the regions that are far away from the non-failure-causing test cases, then these should be more likely to be failure-causing. With these considerations, a family of advanced test case generation methods called *adaptive random testing* (ART) was proposed: ART aims to achieve an even distribution of test cases across the entire input domain [10], [23], [24], and thereby improves on RT across a number of metrics [10].

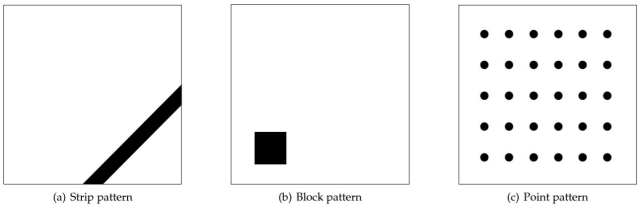

Fig. 1. Three kinds of failure regions in the 2-dimensional input domain [10]

Chan et al. [25] identified and summarized three common types of failure regions: block; point; and strip. Typical examples of these three failure region types in a two-dimensional input domain are shown in Fig. 1. Traditional simulation analysis (e.g., for RT and ART) generally creates one type of artificial failure region based on a pre-defined dimensionality and failure rate, and a failure is detected if the new test case is inside a failure region. The three broad categories of failure regions can be summarized as follows [10], [25]:

- Block failure region: The failure-causing test cases cluster in at least one contiguous area.
- Strip failure region: The failure-causing test cases cluster in a narrow and contiguous linear area.
- Point failure region: The failure-causing test cases are spread in the input domain, individually or in small groups.

Simulations have been used extensively in the development of many software testing techniques [10]. However, to the best of our knowledge, no simulation framework for MT exists. This lack of an applicable framework for MT test simulation motivated some of the work in this paper.

III. A Simulation framework for MT

A. Deterministic Metamorphic Relations

Suppose we are testing a system that is used to calculate the square of a number, $square(x)$. Two possible examples of 1-1 MRs for this system are: (1) $MR_{square-a}$: If a specific positive integer P ($P = 1$) is added to the source input to generate the follow-up input, then the follow-up output should be larger than the source output; and (2) $MR_{square-b}$: If a positive integer Q ($Q > 0$) is added to the source input to generate the follow-up input, then the follow-up output should be larger than the source output. Obviously, for the first kind of 1-1 MR, one STC can generate one and only one FTC, while for the second type of 1-1 MR, given an STC, we can generate different FTCs when using different values for Q. We refer to an MR for which one specific STC generates one (and only) specific FTC as a *Deterministic 1-1 MR* (*DMR[1-1]*). Using this notation, the first MR example above may be written as $DMR[1\text{-}1]_{square-a}$. We generalise this concept to define a *Deterministic M-N MR* (*DMR[M-N]*) as an M-N MR such that for any series of M STCs ($< STC_1, STC_2, \ldots STC_M >$) in a given valid MG, there is only one unique series of N corresponding FTCs ($< FTC_1, FTC_2, \ldots FTC_N >$).

B. Metamorphic Relation Violation Regions

Suppose we are conducting MT on an SUT using a Deterministic 1-1 MR (*DMR[1-1]*): We can use all possible STCs in the input domain, automatically generate the relevant FTCs, and identify the set of those STCs (with their relevant FTCs) that violate the given MR. If we later select an STC from this test set, then this MR will be violated. In this example, we do not need to consider FTCs any further — because the FTCs are fully determined by the STC (and the (D)MR), only the STCs are of interest. The FTC cannot be changed once the STC and the DMR have been decided. At this point, an MT simulation can be created (for this category of MR) where, without any execution of the SUT, violation of this MR can be identified by finding any STC that causes a violation: These STCs are called (D)MR-violating STCs. The MR-violation rate is the number of MR-violating STCs as a proportion of all possible STCs. The *MR-violation region* (MRVR) refers to the regions of MR-violating STCs and MR-violating FTCs. We can further refer to the MRVR-S as the STC-only component of the MRVR — the regions of the input domain from which any STC drawn and used with this MR will result in an MR violation. Similarly, the MRVR-F is the FTC-only component of the MRVR. In this context, determination of whether or not the MR is violated only requires checking if the selected STC is one of the MR-violating STCs (is inside the MRVR-S).

Obviously, the MRVRs will change for different MRs.

For the second type of 1-1 MR, like $MR_{square-b}$, identification of the MRVR-S requires that the value of Q first be set. For instance, when $Q = 1$, we can get one MRVR-S (the same as for $MR_{square-a}$); when $Q = 2$, we may get a new MRVR-S; and so on. These different values of Q can each be considered to create an input domain; or the entire SUT could be considered to have an additional input domain dimension, identified by the different values of P. Further discussion of MRVR identification for other MR types (non-*DMR[1-1]*) is beyond the scope of the present paper, but is part of our ongoing and future work.

A process using DMRs and MRVRs for MT simulations can be summarized as follows:

1) Select a suitable (processed) SUT for which the identified DMR-violation rate and regions are appropriate.
2) If a stopping condition is not triggered, do:
    a) Execute the given (metamorphic) test case generation strategy to get a new STC.
    b) A violation is detected if this new STC is inside an MRVR-S.

C. Relationship between MRVRs and Failure Regions

It is important to emphasise that, for a given faulty SUT, the failure regions and the MRVRs are expected to be different to each other. Obviously, in situations where the SUT faces the oracle problem, there can be no determination of the failure region locations. However, even assuming an oracle, as already noted, MRVRs are specific to a particular MR, with different MRs expected to have different MRVRs, even for the same SUT. Similarly, failure regions are comprised of test cases that reveal a failure, as determined by the oracle. As has been discussed elsewhere [26], many successful testing experiences have included MR violations that revealed faults that could not be identified through traditional oracle-based testing.

For a given faulty SUT with an available oracle and a 1-1 MR, if either the STC or the FTC is in the failure region, then their execution will result in the oracle identifying a failure-causing input (or two, if both the STC and the FTC are in the failure region). However, there is no guarantee that MGs comprising such failure-causing STCs or FTCs would also result in a violation of the MR.

As has been noted in previous studies [2], [27], [28], effective MT often includes the use of several diverse [29] MRs, with their combined ability being able to outperform any individual MR [29], [30]. Although this is perhaps desirable in real testing, the scope of the current paper relates to the provision of simulation tools for specific individual MRs (in particular, Deterministic 1-1 MRs). In this context, the user may prefer to create simulations where MR-violating STCs (MRVR-S) or MR-violating FTCs (MRVR-F) are also failure-revealing (i.e. the MRVR and the failure region are identical). This is, for the MT simulation framework in this paper, a trivial extension.

Related to the failure regions in traditional oracle-based testing or simulations [25], it is reasonable to postulate the existence of similar, MT-oriented, categories of block, point and strip shapes for MRVRs (including at the level of MRVR-S and MRVR-F). For SUTs (in reality or simulations) with Deterministic 1-1 MRs, for example, we can identify: (1) *Block MRVRs*, where the MR-violating STCs/FTCs cluster in at least one contiguous area; (2) *Strip MRVRs*, where the MR-violating STCs/FTCs cluster in a narrow linear area; and (3)

TABLE I
PRELIMINARY CASE STUDY SUTS AND MRS

SUT:	**Sin**	**Bessjy**	**TriSquarePlus**
Dimension of Inputs:	1	2	3
Input Domain:	(-1000, 1000), (0, 1000)	((1, 1), (100, 100))	((0, 0, 0), (100, 100, 100))
MR:	MR_{Sin1}, MR_{Sin2}	MR_{Bessjy}	$MR_{TriSquarePlus}$
Type of MR:	1-1 MR	1-2 MR	1-1 MR
DMR Notation:	$DMR[1\text{-}1]_{Sin1}$, $DMR[1\text{-}1]_{Sin2}$	$DMR[1\text{-}2]_{Bessj}$	$DMR[1\text{-}1]_{TriSquarePlus}$
Number of Mutants:	10	10	10

In the domain of Sin Function
where
- The input contains one parameter, namely x.
- For $DMR[1\text{-}1]_{Sin1}$, the input domain of x is [-1000.0, 1000.0], while that for $DMR[1\text{-}1]_{Sin2}$ is [0.0, 1000.0].
- The output consists of one parameter, represented by S(x).

the following metamorphic relation(s) should hold
- $DMR[1\text{-}1]_{Sin1}$:
  if $x_2 = -x_1$,
  then $S(x_1) = -S(x_2)$.
- $DMR[1\text{-}1]_{Sin2}$:
  if $x_2 = x_1 * 3$,
  then $S(x_1) * 3 = S(x_2) + S^3(x_1) * 4$.

In the domain of Bessjy Function
where
- The input contains two parameters, $1.0 < x < 100.0$, $1.0 < y < 100.0$.
- The output consists of four parameters, represented by $B(x, y)_1$, $B(x, y)_2$, $B(x, y)_3$, $B(x, y)_4$.

the following metamorphic relation(s) should hold
- $DMR[1\text{-}2]_{Bessjy}$:
  if $x_1 = x_2 = x_3$, $y_2 = y_1 - 1$, $y_3 = y_1 + 1$,
  then $2 * y_1 / x_1 * B(x_1, y_1)_1 = B(x_2, y_2)_1 + B(x_3, y_3)_1$.

In the domain of TriSquarePlus Function
where
- The input contains three parameters, $0.0 < x < 100.0$, $0.0 < y < 100.0$, $0.0 < z < 100.0$.
- The output consists of one parameter, represented by T(x, y, z).

the following metamorphic relation(s) should hold
- $DMR[1\text{-}1]_{TriSquarePlus}$:
  if $x_2 = x_1 / 3$, $y_2 = y_1 / 3$, $z_2 = z_1 / 3$,
  then $T(x_1, y_1, z_1) = 9 * T(x_2, y_2, z_2)$.

Fig. 2. Details of the Case Study MRs [7], [11], [31], [32]

Point MRVRs, where the MR-violating STCs/FTCs are spread throughout the input domain, individually or in small groups.

IV. A PRELIMINARY CASE STUDY IDENTIFYING MRVRS FOR DMRS

We conducted a preliminary investigation into the presence of the different types of MRVRs. In the interests of simplicity, and for ease of visualisation and representation, systems with numerical input domains and low dimensionality were selected. The investigation involved examination of four DMRs identified for three pieces of software, all of which have been previously studied and published [7], [11], [31], [32]:

- $DMR[1\text{-}1]_{Sin1}$ and $DMR[1\text{-}1]_{Sin2}$ are 1-1 Deterministic MRs for an implementation of the sine function, $Sin(x)$ [11], [31], [32].
- $DMR[1\text{-}2]_{Bessj}$ is a 1-2 Deterministic MR for an implementation of the Bessel function, $Bessjy(x, y)$ [11], [32].

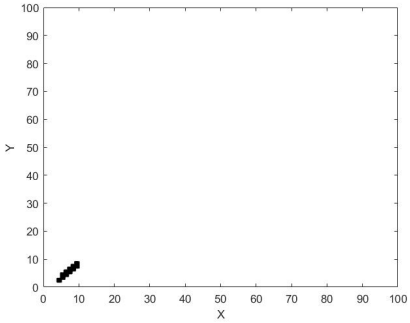

Fig. 3. Block MRVR-S for $Bessjy$ with $DMR[1\text{-}2]_{Bessj}$

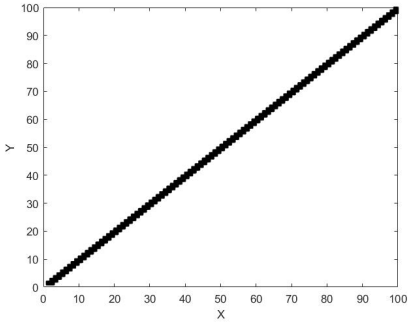

Fig. 4. Strip MRVR-S for $Bessjy$ with $DMR[1\text{-}2]_{Bessj}$

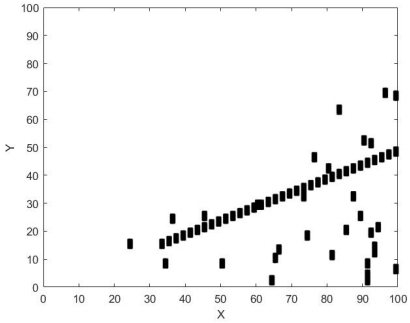

Fig. 5. Point MRVR-S for $Bessjy$ with $DMR[1\text{-}2]_{Bessj}$

- $DMR[1\text{-}1]_{TriSquarePlus}$ is a 1-1 Deterministic MR for an implementation of a program that calculates the type and square of a triangle, $TriSquarePlus(x, y, z)$ [7].

Table I summarises the details of the three SUTs and related (D)MRs; and Fig. 2 presents the MR details.

Ten mutants were created for each SUT. Execution of the STCs (and FTCs) for each mutant SUT allowed for the mapping of the relevant MRVRs. These MRVRs were then manually examined and categorised as one of the three MRVR types. Examples of the three MRVR-S (MRVR for the STCs) types in a 2-dimensional input domain, for the $Bessjy(x, y)$ function, are shown in Figs. 3, 4 and 5:

- The MR-violating STCs in the MRVR-S in Fig. 3 approximately cluster in a contiguous jagged rectangular region,

which we classified as a block MRVR (MRVR-S).
- The MR-violating STCs in the MRVR-S in Fig. 4 approximately cluster in a narrow line connecting adjacent edges, which we classified as a strip MRVR (MRVR-S).
- The MR-violating STCs in the MRVR-S in Fig. 5 are less obviously clustered, and are more diversely spread throughout the input domain, which we classified as a point MRVR (MRVR-S).

Although this was only a preliminary attempt to investigate the possible MRVR types, the results support their presence, and their identification and classification. A more comprehensive and in-depth investigation of this will form part of our future work.

V. Conclusion

As MT has been increasing in popularity, the question of how to (quickly) evaluate its performance, including its efficiency and effectiveness, has become more pressing. Other testing approaches have often benefitted from the use of simulations to enable fast evaluation of testing parameter or configuration settings. The lack of a similar simulation framework for MT has impeded some experimentation and study. In this paper, we have presented a framework to support simulating the MT process. To the best of our knowledge, this is the first such framework in the literature.

Because MT alleviates the need for an oracle, our simulation framework makes use of the concept of MR-violation regions (MRVRs), which can be used for a type of MRs that we have labelled Deterministic MRs (DMRs). For 1-1 DMRs (MRs which have one source test case, and generate exactly one, determined, follow-up test case), the use of the MRVR-S (the MRVR comprising only STCs) allows a simulation to be created where generation of the test case in the MRVR-S can be considered to result in a violation of the MR, without further exploration of the FTC or execution of the SUT.

We discussed the relationship between the oracle-based testing failure regions and MRVRs, noting that they are not expected to be the same regions. Similar to the three categories of failure regions identified in traditional testing [25], we proposed three MRVR shapes/types: block, strip, and point. We also reported on a preliminary case study examining the MRVRs (MRVR-Ss) for some SUTs and MRs from some previous studies, showing the potential to identify these three MRVR types.

A limitation of this paper has been the focus on just one type of DMR. Our future work will involve further MR classifications, and examination of how to prepare related simulations. Other MRVR categories will also be explored. We anticipate that this research direction will enable testers and researchers to more quickly and efficiently explore different MT settings and configurations, leading to insights, and better, and more widespread, use of MT.

VI. Acknowledgments

This work is supported by the Natural Science Foundation of China (project no. 61872167). The authors acknowledge the financial support from the Artificial Intelligence and Optimisation Research Group (AIOP), the Faculty of Science and Engineering (FoSE), and the International Doctoral Innovation Centre. This work was also supported in part by a Western River Entrepreneurship Grant and by Morphick Solutions PTY Ltd, Australia.

References

[1] E. T. Barr, M. Harman, P. McMinn, M. Shahbaz, and S. Yoo, "The oracle problem in software testing: A survey," *IEEE transactions on software engineering*, vol. 41, no. 5, pp. 507–525, 2014.

[2] S. Segura, G. Fraser, A. B. Sanchez, and A. Ruiz-Cortés, "A survey on metamorphic testing," *IEEE Transactions on software engineering*, vol. 42, no. 9, pp. 805–824, 2016. [Online]. Available: http://doi.org//10.1109/TSE.2016.2532875

[3] T. Y. Chen and R. Merkel, "An upper bound on software testing effectiveness," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 17, no. 3, pp. 1–27, 2008.

[4] Z. Q. Zhou, D. Towey, P.-L. Poon, and T. H. Tse, "Introduction to the special issue on test oracles," *Journal of Systems and Software*, vol. 136, p. 187, 2018. [Online]. Available: https://doi.org/10.1016/j.jss.2017.08.031

[5] S. Segura, D. Towey, Z. Q. Zhou, and T. Y. Chen, "Metamorphic testing: Testing the untestable," *IEEE Software*, vol. 37, no. 3, pp. 46–53, 2018.

[6] 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.

[7] Z. Hui, X. Wang, S. Huang, and S. Yang, "MT-ART: A test case generation method based on adaptive random testing and metamorphic relation," *IEEE Transactions on Reliability*, pp. 1–25, 2021.

[8] A. C. Barus, T. Y. Chen, F.-C. Kuo, H. Liu, and H. W. Schmidt, "The impact of source test case selection on the effectiveness of metamorphic testing," in *2016 IEEE/ACM 1st International Workshop on Metamorphic Testing (MET)*. IEEE, 2016, pp. 5–11.

[9] Y. Jia and M. Harman, "An analysis and survey of the development of mutation testing," *IEEE transactions on software engineering*, vol. 37, no. 5, pp. 649–678, 2011. [Online]. Available: http://doi.org//10.1109/TSE.2010.62

[10] R. Huang, W. Sun, Y. Xu, H. Chen, D. Towey, and X. Xia, "A survey on adaptive random testing," *IEEE Transactions on Software Engineering*, pp. 1–32, 2019. [Online]. Available: http://doi.org/10.1109/TSE.2019.2942921

[11] Z. Hui and S. Huang, "MD-ART: A test case generation method without test oracle problem," in *Proceedings of the 1st International Workshop on Specification, Comprehension, Testing, and Debugging of Concurrent Programs*, ser. SCTDCP 2016. New York, NY, USA: Association for Computing Machinery, 2016, pp. 27–34. [Online]. Available: http://doi.org//10.1145/2975954.2975959

[12] T. Y. Chen, F.-C. Kuo, and H. Liu, "On test case distributions of adaptive random testing," in *Proceedings of the Nineteenth International Conference on Software Engineering & Knowledge Engineering (SEKE 2007)*. Knowledge Systems Institute (KSI), 2007, pp. 141–144.

[13] S. Morasca and S. Serra-Capizzano, "On the analytical comparison of testing techniques," in *Proceedings of the 2004 ACM SIGSOFT international symposium on Software testing and analysis*, 2004, pp. 154–164.

[14] ISO/IEC/IEEE, "ISO/IEC/IEEE international standard - systems and software engineering – vocabulary," *ISO/IEC/IEEE 24765:2010(E)*, pp. 1–418, 2010.

[15] A. Orso and G. Rothermel, "Software testing: A research travelogue (2000-2014)," in *Future of Software Engineering Proceedings*, ser. FOSE 2014. New York, NY, USA: Association for Computing Machinery, 2014, pp. 117–132. [Online]. Available: http://doi.org//10.1145/2593882.2593885

[16] P. Bourque and R. E. Fairley, "SWEBOK v3. 0: Guide to the software engineering body of knowledge," *IEEE Computer Society*, pp. 1–335, 2014.

[17] C. Schneckenburger and J. Mayer, "Towards the determination of typical failure patterns," in *Fourth international workshop on Software quality assurance: in conjunction with the 6th ESEC/FSE joint meeting*, 2007, pp. 90–93.

[18] G. B. Finelli, "Nasa software failure characterization experiments," *Reliability Engineering & System Safety*, vol. 32, no. 1-2, pp. 155–169, 1991.

[19] L. J. White and E. I. Cohen, "A domain strategy for computer program testing," *IEEE Transactions on Software Engineering*, no. 3, pp. 247–257, 1980.

[20] P. E. Ammann and J. C. Knight, "Data diversity: An approach to software fault tolerance," *Ieee transactions on computers*, vol. 37, no. 4, pp. 418–425, 1988.

[21] P. G. Bishop, "The variation of software survival time for different operational input profiles (or why you can wait a long time for a big bug to fail)," in *FTCS-23 The Twenty-Third International Symposium on Fault-Tolerant Computing*. IEEE, 1993, pp. 98–107.

[22] T. Y. Chen, F.-C. Kuo, R. G. Merkel, and T. H. Tse, "Adaptive random testing: The ART of test case diversity," *Journal of Systems and Software*, vol. 83, no. 1, pp. 60–66, 2010.

[23] T. Y. Chen, T. H. Tse, and Y.-T. Yu, "Proportional sampling strategy: a compendium and some insights," *Journal of Systems and Software*, vol. 58, no. 1, pp. 65–81, 2001.

[24] T. Y. Chen, H. Leung, and I. K. Mak, "Adaptive random testing," in *Annual Asian Computing Science Conference*. Springer, 2004, pp. 320–329.

[25] F. T. Chan, T. Y. Chen, I. K. Mak, and Y.-T. Yu, "Proportional sampling strategy: guidelines for software testing practitioners," *Information and Software Technology*, vol. 38, no. 12, pp. 775–782, 1996.

[26] T. Y. Chen, F.-C. Kuo, W. Ma, W. Susilo, D. Towey, J. Voas, and Z. Q. Zhou, "Metamorphic testing for cybersecurity," *Computer*, vol. 49, no. 6, pp. 48–55, 2016.

[27] H. Liu, F.-C. Kuo, D. Towey, and T. Y. Chen, "How effectively does metamorphic testing alleviate the oracle problem?" *IEEE Transactions on Software Engineering*, vol. 40, no. 1, pp. 4–22, 2014.

[28] T. Y. Chen, F.-C. Kuo, H. Liu, P.-L. Poon, D. Towey, T. H. Tse, and Z. Q. Zhou, "Metamorphic testing: A review of challenges and opportunities," *ACM Computing Surveys (CSUR)*, vol. 51, no. 1, pp. 1–27, 2018.

[29] T. Y. Chen, F.-C. Kuo, D. Towey, and Z. Q. Zhou, "A revisit of three studies related to random testing," *Science China Information Sciences*, vol. 58, pp. 052 104:1–052 104:9, 2015. [Online]. Available: https://doi.org/10.1007/s11432-015-5314-x

[30] Z. Q. Zhou, J. Zhu, T. Y. Chen, and D. Towey, "In-place metamorphic exploration and testing," in *2022 IEEE/ACM 7th International Workshop on Metamorphic Testing (MET)*. IEEE, 2022, [Accepted to appear].

[31] G. Dong, "Metamorphic testing techniques for error detection efficiency," Ph.D. dissertation, Nanjing, China: School of Computer Science and Engineering, Southeast University, 2009.

[32] T. Y. Chen, F.-C. Kuo, Y. Liu, and A. Tang, "Metamorphic testing and testing with special values," in *Proceedings of the 5th International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing (SNPD'04)*, 2004, pp. 128–134.