A Survey on Metamorphic Testing

Sergio Segura, *Member, IEEE*, Gordon Fraser, *Member, IEEE*, Ana B. Sanchez, and Antonio Ruiz-Cortés

Abstract—A test oracle determines whether a test execution reveals a fault, often by comparing the observed program output to the expected output. This is not always practical, for example when a program's input-output relation is complex and difficult to capture formally. *Metamorphic testing* provides an alternative, where correctness is not determined by checking an individual concrete output, but by applying a transformation to a test input and observing how the program output "morphs" into a different one as a result. Since the introduction of such *metamorphic relations* in 1998, many contributions on metamorphic testing have been made, and the technique has seen successful applications in a variety of domains, ranging from web services to computer graphics. This article provides a comprehensive survey on metamorphic testing: It summarises the research results and application areas, and analyses common practice in empirical studies of metamorphic testing as well as the main open challenges.

Index Terms—Metamorphic testing, oracle problem, survey

1 Introduction

Software testing is an essential but costly activity applied during software development to detect faults in programs. Testing consists of executing a program with test inputs, and to detect faults there needs to be some procedure by which testers can decide whether the output of the program is correct or not, a so-called *test oracle* [1]. Often, the test oracle consists of comparing an expected output value with the observed output, but this may not always be feasible. For example, consider programs that produce complex output, like complicated numerical simulations, or code generated by a compiler—predicting the correct output for a given input and then comparing it with the observed output may be non-trivial and error-prone. This problem is referred to as the *oracle problem* and it is recognised as one of the fundamental challenges of software testing [1], [2], [3], [4].

Metamorphic testing [5] is a technique conceived to alleviate the oracle problem. It is based on the idea that often it is simpler to reason about relations between outputs of a program, than it is to fully understand or formalise its input-output behaviour. The prototypical example is that of a program that computes the sine function: What is the exact value of $\sin(12)$? Is an observed output of -0.5365 correct? A mathematical property of the sine function states that $\sin(x) = \sin(\pi - x)$, and we can use this to test whether $\sin(12) = \sin(\pi - 12)$ without knowing the concrete values of either sine calculation. This is an example of a *metamorphic relation*: an input transformation that can be used to generate new test cases from existing test data, and an output relation, that compares the outputs produced by a pair of test cases. Metamorphic testing does not only alleviate the oracle problem, but it can also be highly automated.

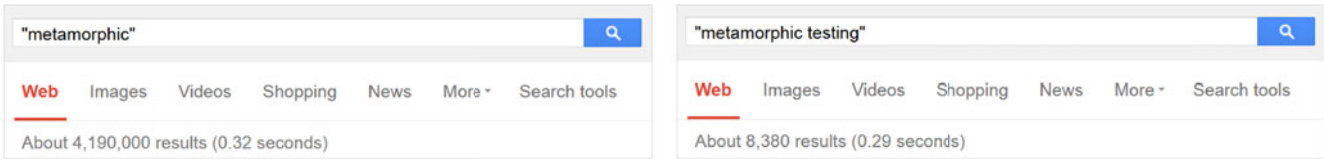
The introduction of metamorphic testing can be traced back to a technical report by Chen et al. [5] published in 1998. However, the use of identity relations to check program outputs can be found in earlier articles on testing of numerical programs [6], [7] and fault tolerance [8]. Since its introduction, the literature on metamorphic testing has flourished with numerous techniques, applications and assessment studies that have not been fully reviewed until now. Although some papers present overviews of metamorphic testing, they are usually the result of the authors' own experience [9], [10], [11], [12], [13], review of selected articles [14], [15], [16] or surveys on related testing topics [3]. At the time of writing this article, the only known survey on metamorphic testing is written in Chinese and was published in 2009[1] [17]. As a result, publications on metamorphic testing remain scattered in the literature, and this hinders the analysis of the state of the art and the identification of new research directions.

In this article, we present an exhaustive survey on metamorphic testing, covering 119 papers published between 1998 and 2015. To provide researchers and practitioners with an entry point, Section 2 contains an introduction to metamorphic testing. All papers were carefully reviewed and classified, and the review methodology followed in our survey as well as a brief summary and analysis of the selected papers are detailed in Section 3. We summarise the state of the art by capturing the main advances on metamorphic testing in Section 4. Across all surveyed papers, we identified more than 12 different application areas, ranging from web services through simulation and modelling to computer graphics (Section 5). Of particular interest for researchers is a detailed analysis of experimental studies and evaluation metrics (Section 6). As a result of our survey, a number of research challenges emerge, providing avenues for future research (Section 7); in particular, there are open questions on how to derive effective metamorphic relations, as well as how to reduce the costs of testing with them.

- *S. Segura, A.B. Sánchez, and A. Ruiz-Cortés are with the Department of Computer Languages and Systems, Universidad de Sevilla, Spain. E-mail: {sergiosegura, anabsanchez, aruiz}@us.es.*
- *G. Fraser is with the Department of Computer Science, University of Sheffield, Sheffield, United Kingdom. E-mail: gordon.fraser@sheffield.ac.uk.*

Manuscript received 11 July 2015; revised 9 Feb. 2016; accepted 14 Feb. 2016. Date of publication 28 Feb. 2016; date of current version 23 Sept. 2016. Recommended for acceptance by P. Tonella.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TSE.2016.2532875

1. Note that 86 out of the 119 papers reviewed in our survey were published in 2009 or later.



(a) Source test case. (b) Follow-up test case.

Fig. 1. Metamorphic test on the Google search engine checking the relation $Count(q) \geq Count(q+k)$.

2 Metamorphic Testing

Identity relations are a well-known concept in testing, and have been used even before the introduction of metamorphic relations. For example, Blum et al. [7] checked whether numerical programs satisfy identity relations such as $P(x) = P(x_1) + P(x_2)$ for random values of x_1 and x_2. In the context of fault tolerance, the technique of data diversity [8] runs the program on re-expressed forms of the original input; e.g., $\sin(x) = \sin(a) \times \sin(\pi/2 - b) + \sin(\pi/2 - a) \times \sin(b)$ where $a + b = x$. The concept of metamorphic testing, introduced by Chen [5] in 1998, generalises these ideas from identity relations to any type of relation, such as equalities, inequalities, periodicity properties, convergence constraints, subsumption relationships and many others. In general, a metamorphic relation for a function f is expressed as a relation among a series of function inputs $x_1, x_2, \ldots, x_n$ (with $n > 1$), and their corresponding output values $f(x_1), f(x_2), \ldots, f(x_n)$ [18]. For instance, for the sine example from the introduction the relation between x_1 and x_2 would be $\pi - x_1 = x_2$, and the relation between $f(x_1)$ and $f(x_2)$ would be equality, i.e.,:

$$R = \{(x_1, x_2, \sin x_1, \sin x_2) \mid \pi - x_1 = x_2 \rightarrow \sin x_1 = \sin x_2\}.$$

This resembles the traditional concept of program invariants, which are properties (for example expressed as assert statements) that hold at certain points in programs [19]. However, the key difference is that an invariant has to hold for every possible program execution, whereas a metamorphic relation is a relation between *different* executions. A relation between two executions implicitly defines how, given an existing source test case (x_1), one has to transform this into a follow-up test case (x_2), such that an abstract relation R (e.g., $\sin x_1 = \sin x_2$) can be checked on the inputs represented by x_1 and x_2, as well as the outputs produced by executing x_1 and x_2. The term metamorphic relation presumably refers to this "metamorphosis" of test inputs and outputs. If the relation R does not hold on a pair of source and follow-up test cases x_1 and x_2, then a fault has been detected. In this article, we use the term *metamorphic test case* to refer to a pair of a source test case and its follow-up test case.

The basic process for the application of metamorphic testing can be summarised as follows:

1) *Construction of metamorphic relations.* Identify necessary properties of the program under test and represent them as metamorphic relations among multiple test case inputs and their expected outputs, together with some method to generate a follow-up test case based on a source test case. Note that metamorphic relations may be associated with preconditions that restrict the source test cases to which they can be applied.
2) *Generation of source test cases.* Generate or select a set of source test cases for the program under test using any traditional testing technique (e.g., random testing).
3) *Execution of metamorphic test cases.* Use the metamorphic relations to generate follow-up test cases, execute source and follow-up test cases, and check the relations. If the outputs of a source test case and its follow-up test case violate the metamorphic relation, the metamorphic test case is said to have failed, indicating that the program under test contains a bug.

As an illustrative example, consider a program that computes the shortest path between a source vertex s and destination vertex d in a graph G, $SP(G, s, d)$. A metamorphic relation of the program is that if the source and destination vertices are swapped, the length of the shortest path should be equal: $|SP(G, s, d)| = |SP(G, d, s)|$. Suppose that a source test case (G, a, b) is selected according to some testing method (e.g., randomly). Based on the metamorphic relation, we can now easily generate a new follow–up test case by swapping the source and destination vertices (G, b, a). After executing the program with both test cases, their outputs can be checked against the relation to confirm whether it is satisfied or not, i.e., whether the outputs are equal. If the metamorphic relation is violated, it can be concluded that the metamorphic test has failed and the program is faulty.

As a further example, consider testing an online search engine such as Google or Yahoo [20]. Let $Count(q)$ be the number of results returned for a search query q. Intuitively, the number of returned results for q should be greater or equal than that obtained when refining the search with another keyword k. This can be expressed as the following metamorphic relation: $Count(q) \geq Count(q + k)$, where $+$ denotes the concatenation of two keywords. Fig. 1 illustrates the application of this metamorphic relation on Google. Consider a source test case consisting in a search for the keyword "metamorphic", resulting in "*About*" 4.2M results. Suppose that a follow-up test case is constructed by searching for the keywords "metamorphic testing": This leads to 8,380 results which is less than the result for "metamorphic", and thus satisfies the relation. If more results were found, then that would violate the metamorphic relation, revealing a bug in the system.

If source test cases are generated automatically, then metamorphic testing enables full test automation, i.e., input generation and output checking. In the sine example presented in Section 1, for instance, metamorphic testing could be used together with random testing to automatically generate random source test cases (x) and their respective follow-up test cases ($\pi - x$), until a pair is found that violates the metamorphic relation, or a maximum time out is

reached. Similarly, in the search engine example, metamorphic testing could also be used together with a random word generator to automatically construct source test cases (e.g., "algorithm") and their respective follow-up test cases (e.g., "algorithm colour") until a pair that reveals a bug is found, if any such pairs exists.

3 Review Method

To perform a survey on metamorphic testing we followed a systematic and structured method inspired by the guidelines of Kitchenham [21] and Webster et al. [22]. A similar approach was followed by some of the authors in the context of software product lines [23]. To report the results, we also took inspiration from recent surveys on related topics such as the oracle problem [3], search-based testing [24], automated test case generation [2] and mutation analysis [25]. Below, we detail the main data regarding the review process and its results.

3.1 Research Questions

The aim of this survey is to answer the following research questions on metamorphic testing:

*RQ*1: *What improvements to the technique have been made?*
*RQ*2: *What are its known application domains?*
*RQ*3: *How are experimental evaluations performed?*
*RQ*4: *What are the future research challenges?*

We propose RQ1 to obtain an in-depth view on metamorphic testing outlining the state of the art in terms of the main advances in the application of the technique since its original introduction. RQ2 is proposed to give an insight into the scope of metamorphic testing and its applicability to different domains including its integration with other testing techniques. We also want to know how different approaches of performing metamorphic testing are evaluated including the subject programs used, types of detected faults, evaluation metrics, and empirical studies involving humans. Finally, based on the answer to the previous questions, we expect to identify unresolved problems and research opportunities in response to RQ4.

3.2 Inclusion and Exclusion Criteria

We scrutinised the existing literature, looking for papers addressing any topic related to metamorphic testing, including methods, tools or guidelines for the application of the technique, applications to specific testing problems, empirical evaluations, and surveys. Articles of the same authors but with very similar content were intentionally classified and evaluated as separate contributions for a more rigorous analysis. Later, in the presentation of results, we grouped those articles with no major differences. We excluded PhD theses as well as those papers not related to the computer sciences field, not written in English, or not accessible on the Web.

3.3 Source Material and Search Strategy

The search for relevant papers was carried out in the online repositories of the main technical publishers, including ACM, Elsevier, IEEE, Springer and Wiley. We collected computer science papers published between January 1st 1998 (when Chen's report was published) and November 30th 2015 which have either "metamorphic test," "metamorphic testing," "metamorphic relation" or "metamorphic relations" in their title, abstract or keywords. We refer the reader to the technical report [26] which contains all data forms and query details. After a quick review of the results, we noticed that some articles on metamorphic testing with many citations were not among the candidate papers, including the technical report of Chen et al. [5] where the technique was introduced. To include those papers, we performed the search in the Google Scholar database, and additionally selected all papers with 5 or more citations published outside our target publication sources[2]. These were merged with our previous results, resulting in a final set of 362 candidate papers.

TABLE 1
Search Engines and Number of Primary Studies

Search engine	Primary studies
ACM digital library	15
Elsevier ScienceDirect	6
IEEEXplore digital library	65
Springer online library	13
Wiley InterScience	4
Google Scholar (+5 citations)	16
Total	**119**

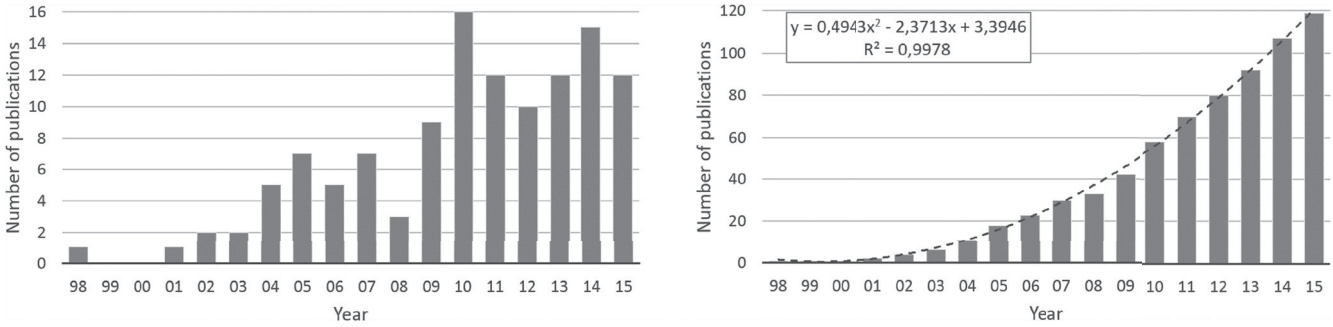
Next, we examined the abstracts of the papers identified in the previous step and filtered them according to our inclusion and exclusion criteria, checking the content of the papers when unsure. This step was performed by two different authors who agreed on the results. The set of candidate papers was filtered to 116 publications within the scope of our survey. Then, we contacted the corresponding authors of the 116 selected papers and asked them to inform us about any missing papers within the scope of our search. Based on the feedback received, we included 3 new papers meeting our search criteria, except for the inclusion of the search terms in their title, abstract or keywords. As a result, the search was finally narrowed to *119 publications* that were in the scope of this survey. These papers are referred to as the *primary studies* [21]. Table 1 presents the number of primary studies retrieved from each source.

It is possible that our search has failed to find all papers since we focused on a subset of reputed publishers. However, we remain confident that the overall trends we report are accurate and provide a fair picture of the state of the art on metamorphic testing.

3.4 Data Collection

All 119 primary studies were carefully analysed to answer our research questions. For each study, we extracted the following information: full reference, brief summary, type of contribution (e.g., case study), application domains, integration with other testing techniques, number of metamorphic relations proposed, evaluation details, lessons learned and suggested challenges. To facilitate the process, we filled in a data extraction form for each primary study. All data forms were collected and published in a technical report [26].

2. The search was performed on December 30th, 2015.



(a) Number of publications per year.

(b) Cumulative number of publications per year.

Fig. 2. Metamorphic testing papers published between January 1st 1998 and November 30th 2015.

Primary studies were read at least twice by two different authors to reduce misunderstandings or missing information. As a sanity check, we contacted the corresponding author of each primary study and sent them the technical report to confirm that the information collected from their papers was correct. Some minor changes were proposed and corrected. We also asked them to inform us about any missing paper within the scope of our search as described in the previous section.

3.5 Summary of Results

The following sections summarise the primary studies in terms of publication trends, authors, venues, and research topics on metamorphic testing.

3.5.1 Publication Trends

Fig. 2a illustrates the number of publications on metamorphic testing published between January 1st 1998 and November 30th 2015. The graph shows a constant flow of papers on the topic since 2001, in particular from 2010 onwards. The cumulative number of publications is illustrated in Fig. 2b. We found a close fit to a quadratic function with a high determination coefficient ($R^2 = 0.997$), indicating a strong polynomial growth, a sign of continued health and interest in the subject. If the trend continues, there will be more than 170 metamorphic testing papers by 2018, two decades after the introduction of the technique.

3.5.2 Researchers and Organisations

We identified 183 distinct co-authors from 74 different organisations in the 119 primary studies under review. Table 2 presents the top authors on metamorphic testing and their most recent affiliation. Unsurprisingly, Prof. T. Y. Chen, with 44 papers, is the most prolific author on the topic.

3.5.3 Geographical Distribution of Publications

We related the geographical origin of each primary study to the affiliation country of its first co-author. Interestingly, we found that all 119 primary studies originated from only 11 different countries with Australia and China ahead, as presented in Table 3. By continents, 37 percent of the papers originated from Asia, 30 percent from Oceania, 19 percent from Europe and 14 percent from America. This suggests that the metamorphic testing community is formed by a modest number of countries but fairly distributed around the world.

3.5.4 Publication Venues

The 119 primary studies under review were published in 72 distinct venues. This means that the metamorphic testing literature is very dispersed, probably due to its applicability to multiple testing domains. Regarding the type of venue, most papers were presented at conferences and symposia (58 percent), followed by journals (23 percent), workshops (16 percent) and technical reports (3 percent). Table 4 lists the venues where at least three metamorphic testing papers have been presented.

3.5.5 Types of Contributions and Research Topics

Fig. 3a classifies the primary studies according to the type of contribution. We found that half of the papers present case

TABLE 2
Top 10 Co-Authors on Metamorphic Testing

Author	Institution	Papers
T. Y. Chen	Swinburne University of Technology	44
T. H. Tse	The University of Hong Kong	20
F.-C. Kuo	Swinburne University of Technology	17
Z. Q. Zhou	University of Wollongong	14
W. K. Chan	City University of Hong Kong	11
H. Liu	RMIT University	9
C. Murphy	Columbia University	9
G. Kaiser	Columbia University	8
X. Xie	Swinburne University of Technology	7
B. Xu	Nanjing University	7

TABLE 3
Geographical Distribution of Publications

Country	Papers
Australia	36
China	25
United States	17
Hong Kong	12
Germany	8
Spain	7
India	5
United Kingdom	3
Switzerland	3
Malaysia	2
France	1

TABLE 4
Top Venues on Metamorphic Testing

Venue	Papers
Int Conference on Quality Software	9
Int Computer Software & Applications Conference	8
Int Workshop on Automation of Software Test	4
Int Conference on Software Engineering	4
IEEE Transactions on Software Engineering	4
Software Testing, Verification and Reliability	4
Int Conf on Software Testing, Verification and Validation	3
Information and Software Technology	3

studies (50 percent), followed by new techniques and methodologies (31 percent), and assessments and empirical studies (10 percent). We also found a miscellany of papers (7 percent) including related surveys, tutorial synopsis, and guidelines. Only two of the papers (2 percent) presented a tool as their main contribution.

A similar classification based on the main research topic is presented in Fig. 3b. Interestingly, we found that 49 percent of the papers report applications of metamorphic testing to different problem domains. The rest of papers address the construction of metamorphic relations (19 percent), integration with other testing techniques (10 percent), assessment of metamorphic testing (6 percent), execution of metamorphic test cases (5 percent) and generation of source test cases (4 percent). Finally, a few papers (7 percent) present brief overviews on the technique, its applications and research directions.

4 State of the Art in Metamorphic Testing

In this section, we address RQ1 by summarising the main contributions to metamorphic testing in the literature. First, we review the papers studying the properties of effective metamorphic relations. Then, approaches are classified according to the step they contribute to in the metamorphic testing process presented in Section 2, namely, construction of metamorphic relations, generation of source test cases, and execution of metamorphic test cases.

4.1 Properties of Good Metamorphic Relations

The effectiveness of metamorphic testing is highly dependent on the specific metamorphic relations that are used, and designing effective metamorphic relations is thus a critical step when applying metamorphic testing. For most problems, a variety of metamorphic relations with different fault-detection capability can be identified [9], [16], [18], [27], [28], [29], [30], [31], [32], [33], [34], [35]. Therefore, it is advisable to use a variety of diverse metamorphic relations to effectively test a given program. Several authors even suggest using as many metamorphic relations as possible during testing [28], [29], [36], [37]. However, because defining metamorphic relations can be difficult, it is important to know how to select the most effective ones. In this section, we review papers studying the properties that make metamorphic relations *good* at detecting faults.

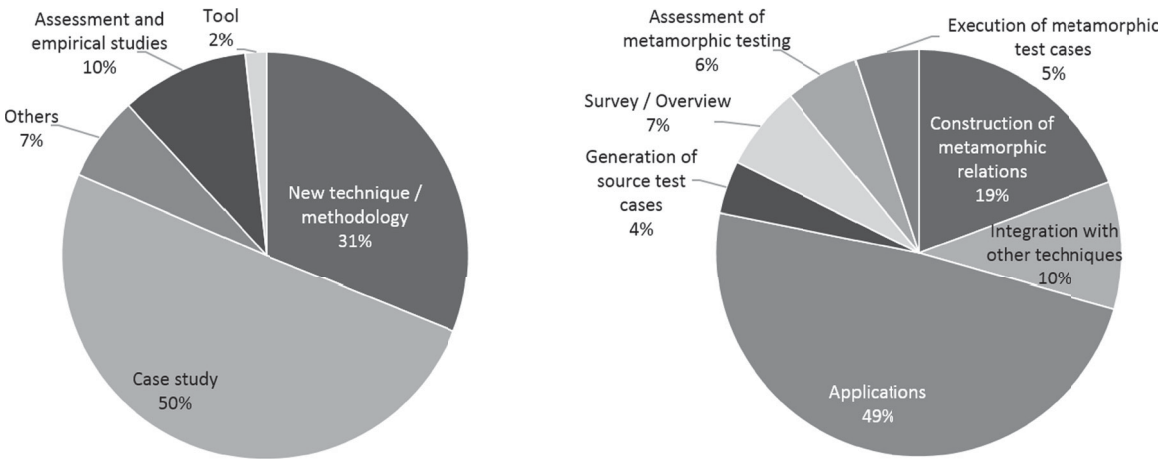
Defining good metamorphic relations requires knowledge of the problem domain. Chen et al. [27] compared the effectiveness of metamorphic relations solely based on the theoretical knowledge of the problem (black-box) versus those derived from the program structure (white-box) using two case studies. They concluded that theoretical knowledge of the problem domain is not adequate for distinguishing good metamorphic relations. Instead, good metamorphic relations should be preferably selected with regard to the algorithm under test following a white-box approach. However, this was later disputed by Mayer and Guderlei [38], who studied six subject programs for matrix determinant computation with seeded faults. They concluded that metamorphic relations in the form of equalities or linear equations[3] as well as those close to the implementation strategy have limited effectiveness. Conversely, they reported that good metamorphic relations are usually strongly inspired by the semantics of the program under test. Other studies have also emphasised the knowledge of the problem domain as a requirement for the application of metamorphic testing [30], [39], [40].

Metamorphic relations should make execution of the follow-up test case as different as possible from the source test case. Chen et al. [27] reported that good metamorphic relations are those that can make the execution of the source-test case as different as possible to its follow-up test case. They defined the "difference among executions" as any aspects of program runs (e.g., paths traversed). This observation has been confirmed by several later studies [9], [41], [42], [43], [44], [45]. In particular, Asrafi et al. [46] hypothesised that the higher the combined code coverage of the source and follow-up test cases, the more different are the executions, and the more effective is the metamorphic relation. Their study on two subject programs showed a strong correlation between coverage and fault-detection effectiveness in one of the two. In a similar study, Cao et al. [47] assessed the relation between fault-detection effectiveness of metamorphic relations and test case dissimilarity. An extensive experiment with 83 faulty programs and 7 distance metrics between the execution profiles of source and follow-up test cases revealed a strong and statistically significant correlation between the fault-detection capability of metamorphic relations and the distance among test cases, in particular when using branch coverage Manhattan distance [48].

Metamorphic relations derived from specific parts of the system are more effective than those targeting the whole system. Several authors have explored the applicability of metamorphic testing for integration testing with some helpful conclusions for the construction of good metamorphic relations. Just and Schweiggert [49], [50] assessed the applicability of metamorphic testing for system and integration testing in the context of an image encoder. Among other results, they concluded that the metamorphic relations derived from the components of a system are usually better at detecting faults than those metamorphic relations derived from the whole system. This finding was later confirmed by Xie et al. [51], who reported that metamorphic relations targeting specific parts of the program under test are easier to construct, more constrained, and therefore more effective in detecting faults than metamorphic relations at the system level.

Metamorphic relations should be formally described. Chan et al. [52] formally described metamorphic relations and

3. The authors literally refer to "equations with linear combinations on each side (with at least two terms on one of the sides)".



(a) Type of contribution. (b) Research topic.

Fig. 3. Classification of primary studies by publication type and research topic.

metamorphic testing for a precise definition of the technique. Their formalisation was reused by several authors [29], [36] and later revised by Chan and Tse [12]. Hui and Huang [53] pointed out that most metamorphic relations in the literature are informally described using natural language, which makes them easily misunderstood, ambiguous and hard to reuse. The authors suggested that good metamorphic relations should be formally described and proposed a formal model for the rigorous description of metamorphic relations using predicate logic, inspired by the work of Chan et al. [52]. In particular, they proposed representing a metamorphic relation as a 3-tuple composed of i) relation between the inputs of source and follow-up test cases, ii) relation between the outputs of source and follow-up test cases, and iii) program function.

4.2 Construction of Metamorphic Relations

Constructing metamorphic relations is typically a manual task that demands thorough knowledge of the program under test. In this section, we review proposed alternative ways to create metamorphic relations, either by combining existing relations, or by generating them automatically.

Liu et al. [54] proposed a method named *Composition of Metamorphic Relations (CMR)* to construct new metamorphic relations by combining several existing relations. A similar idea had been superficially explored previously by Dong et al. [55]. The rationale behind this method is that the resulting relations should embed all properties of the original metamorphic relations, and thus they should provide similar effectiveness with a fewer number of metamorphic relations and test executions. Intuitively, Liu et al. defined two metamorphic relations as *"compositable"* if the follow-up test cases of one of the relations can always be used as source test case of the other. The composition is sensitive to the order of metamorphic relations and generalisable to any number of them. Determining whether two metamorphic relations are composable is a manual task. The results of a case study with a bioinformatics program processing an input matrix show that the composition of a set of metamorphic relations usually produces a composite relation with higher (or at least similar) fault-detection effectiveness than the original metamorphic relations, provided that all component relations have similar "tightness." The tightness of a relation determines how hard it is to satisfy it by mere chance—the tighter a relation is, the more difficult it is to satisfy it with some random outputs; e.g., $\sin(x) = \sin(\pi - x)$ is tighter than $\sin(x) \neq \sin(\pi - x/2)$. They also concluded that the CMR method delivers higher cost-effectiveness than classic metamorphic testing since it involves fewer test executions.

Kanewala and Bieman [56], [57] proposed a method that determines, given a predefined set of relations that they believe to hold for many numerical programs, which of these are exhibited by a given numerical program. Their method works by extracting a function's control flow graph and building a predictive model using machine learning techniques; i.e., it is a white-box method that requires static access to the source code. The approach was evaluated by constructing a prediction model using a code corpus of 48 mathematical functions with numerical inputs and outputs. The model was designed to predict three specific types of metamorphic relations: permutative, additive and inclusive [58]. In addition, they checked the fault–detection effectiveness of the predictive metamorphic relations using seeded faults. The results revealed that 66 percent of the faults (655 out of 988) were detected by the predicted metamorphic relations. In later work [59], the authors extended their method using graph kernels, which provide various ways of measuring similarity among graphs. The intuition behind their approach was that functions that have similar control flow and data dependency graphs may have similar metamorphic relations. Empirical results on the prediction of six different types of metamorphic relations on a corpus of 100 numerical programs revealed that graph kernels lead to higher prediction accuracy.

Zhang et al. [60] proposed a search-based approach for the inference of polynomial metamorphic relations. More specifically, the algorithm searches for metamorphic relations in the form of linear or quadratic equations (e.g., $\cos(2x) = 2\cos^2(x) - 1$). Relations are inferred by running the program under test repeatedly, searching for relations among the inputs and outputs. It is therefore a black-box approach which requires no access to the source code. Since running the program with all the possible input values is rarely possible, the relations identified are strictly referred

to as *likely* metamorphic relations, until they are confirmed by a domain expert. Their work was evaluated inferring hundreds of likely metamorphic relations for 189 functions of 4 commercial and open source mathematical libraries. The results showed that the generated metamorphic relations are effective in detecting mutants. Notice that in contrast to the work of Kanewala and Bieman [56], [57], this approach does not predict whether the program exhibits a previously defined metamorphic relation, but rather infers the metamorphic relation from scratch.

Carzinaga et al. [61] proposed to generate oracles by exploiting the redundancy contained in programs. Given a source test case, they generate a test with the same code in which some operations are replaced with redundant ones. For instance, in the `AbstractMultimap<K,V>` class of the Google Guava library,[4] the methods `put(k,v)` and `putAll(k,c)` are equivalent when `c` is a collection containing a single element `v`. If the outputs of both test cases are not equal, the code must contain a bug. The author presented an implementation of their approach using aspects. The identification of redundant methods is a manual task. Although the core of their contribution was not related to metamorphic testing, their approach can be considered a specific application of the technique. In a related article, Goffi et al. [62], [63] presented a search-based algorithm for the automated synthesis of *likely-equivalent* method sequences in object-oriented programs. The authors suggest that such likely-equivalent sequences could be used as metamorphic relations during testing. The approach was evaluated using 47 methods of 7 classes taken from the Stack Java Standard Library and the Graphstream library. The algorithm automatically synthesised 87 percent (123 out of 141) of the equivalent method sequences manually identified.

Su et al. [64] presented an approach named KABU for the dynamic inference of likely metamorphic relations inspired by previous work on the inference of program invariants [19]. The inference process is constrained by searching for a set of predefined metamorphic relations [58]. A Java tool implementing the approach was presented and evaluated on the inference of likely metamorphic relations in two sample programs. As a result, KABU found more likely metamorphic relations than a group of 23 students trained in the task. Authors also proposed a method, *Metamorphic Differential Testing (MDT)*, built upon KABU, to compare the metamorphic relations between different versions of the same program reporting the differences as potential bugs. Experimental results on different versions of two classification algorithms showed that MDT successfully detected the changes reported in the logs of the Weka library.

Chen et al. [65] presented a specification-based methodology and associated tool called METRIC for the identification of metamorphic relations based on the category-choice framework [66]. In this framework, the program specification is used to partition the input domain in terms of categories, choices and complete test frames. Roughly speaking, a complete test frame is an abstract test case defining possible combinations of inputs, e.g., *{type of vehicle, weekday, parking hours}*. Given a set of complete test frames, METRIC guides testers on the identification of metamorphic relations and related source and follow-up test cases. The results of an empirical study with 19 participants suggest that METRIC is effective and efficient at identifying metamorphic relations.

4.3 Generation of Source Test Cases

As mentioned in Section 6.2, most contributions on metamorphic testing use either random test data or existing test suites for the creation of source test cases. In this section, we review the papers proposing alternative methods for the generation of source test cases.

Gotlieb and Botella [67] presented a framework named *Automated Metamorphic Testing (AMT)* to automatically generate test data for metamorphic relations. Given the source code of a program written in a subset of C and a metamorphic relation, AMT tries to find test cases that violate the relation. The underlying method is based on the translation of the code into an equivalent constraint logic program over finite domains. The solving process is executed until a solution is found or a timeout is reached. The supported types of metamorphic relations are limited to numeric expressions over integers. The framework was evaluated using three laboratory programs with seeded faults.

Chen et al. [28] compared the effectiveness of "special values" and random testing as source test cases for metamorphic testing. Special values are test inputs for which the expected output is well known (e.g., $\sin(\pi/2) = 1$). Since test cases with special values must be manually constructed we consider them as manual testing. The authors found that manual and metamorphic testing are complementary techniques, but they also note that random testing has the advantage of being able to provide much larger test data sets. In a closely related study, Wu et al. [68] concluded that random source test cases result in more effective metamorphic test cases than those derived from manual test cases (special values). Segura et al. [69] compared the effectiveness of random testing and a manually designed test suite as the source test cases for metamorphic testing, and their results also showed that random source test cases are more effective at detecting faults than manually designed source test cases in all the subject programs. Even though this suggests that random testing is more effective, there are also indications that *combining* random testing with manual tests may be even better: Chen et al [28] concluded that random testing is an efficient mechanism to augment the number of source test cases; Segura et al. [69] observed that combining manual tests with random tests leads to faster fault detection compared to using random tests only.

Batra and Sengupta [41] presented a genetic algorithm for the selection of source test cases maximising the paths traversed in the program under test. The goal is to generate a small but highly effective set of source test cases. Their algorithm was evaluated by generating source test cases for several metamorphic relations in a small C program, which determines the type of a triangle, where 4 mutants were generated and killed. In related work, Chen et al. [42] addressed the same problem from a black-box perspective. They proposed partitioning the input domain of the program under test into equivalence classes, in which the program is expected to process the inputs in a similar way. Then, they proposed an algorithm to select test cases that cover those equivalence classes. Evaluation on the triangle

4. https://github.com/google/guava.

program suggests that their algorithm can generate a small set of test cases with high detection rate.

Dong and Zhang [44] presented a method for the construction of metamorphic relations and their corresponding source test cases using symbolic execution. The method first analyses the source code of the program to determine the symbolic inputs that cause the execution of each path. Then, the symbolic inputs are manually inspected and used to guide the construction of metamorphic relations that can exercise all the paths of the program. Finally, source test cases are generated by replacing the symbolic inputs by real values. As in previous work, the approach was evaluated using a small C program with seeded faults.

4.4 Execution of Metamorphic Test Cases

The execution of a metamorphic test case is typically performed in two steps. First, a follow-up test case is generated by applying a transformation to the inputs of a source test case. Second, source and follow-up test cases are executed, checking whether their outputs violate the metamorphic relation. In this section, we present those articles that either propose a different approach for the execution of metamorphic test cases, or to automate part of the process.

Several papers have contributed to the execution and assessment of metamorphic test cases. Wu [70] presented a method named *Iterative Metamorphic Testing (IMT)* to systematically exploit more information from metamorphic tests, by applying metamorphic relations iteratively. In IMT, a sequence of metamorphic relations are applied in a chain style, by reusing the follow-up test case of each metamorphic relation as the source test case of the next metamorphic relation. A case study was presented with a program for sparse matrix multiplication and more than 1,300 mutants. The results revealed that IMT detects more faults than classic metamorphic testing and special value testing. Dong et al. [71] presented an algorithm integrating IMT and program path analysis. The algorithm runs metamorphic tests iteratively until a certain path coverage criterion is satisfied. Segura et al. [69], [72], [73] presented a metamorphic testing approach for the detection of faults in variability analysis tools. Their method is based on the iterative application of a small set of metamorphic relations. Each relation relates two input variability models and their corresponding set of configurations, (i.e., output). In practice, the process can generate an unlimited number of random test cases of any size. In certain domains, it was necessary to apply the metamorphic relations in a certain order. Their approach was proven effective in detecting 19 real bugs in 7 different tools.

Guderlei and Mayer [74] proposed *Statistical Metamorphic Testing (SMT)* for the application of metamorphic testing to non-deterministic programs. SMT does not consider a single execution, but is based on studying the statistical properties of multiple invocations to the program under test. The method works by generating two or more sequences of outputs by executing source and follow-up test cases. Then, the sequences of outputs are compared according to their statistical properties using statistical hypothesis tests. The applicability of the approach was illustrated with a single metamorphic relation on a subject program with seeded faults. In later work, Murphy et al. [75], [76] successfully applied SMT to the detection of faults in a health care simulation program with non-deterministic time events.

Murphy et al. [77], [78] presented an extension of the Java Modelling Language (JML) [79] for the specification and runtime checking of metamorphic relations. Their approach extends the JML syntax to enable the specification of metamorphic properties, which are included in the Java source code as annotations. The extension was designed so it could express the typical metamorphic relations observed by the authors in the domain of machine learning [80]. Additionally, they presented a tool, named Corduroy, that pre-processes the specification of metamorphic relations and generates test code that can be executed using JML runtime assertion checking, ensuring that the relations hold during program execution. For the evaluation, they specified 25 metamorphic relations on several machine learning applications uncovering a few defects.

Murphy et al. [81] presented a framework named Amsterdam for the automated application of metamorphic testing. The tool takes as inputs the program under test and a set of metamorphic relations, defined in an XML file. Then, Amsterdam automatically runs the program, applies the metamorphic relations and checks the results. The authors argue that in certain cases slight variations in the outputs are not actually indicative of errors, e.g., floating point calculations. To address this issue, the authors propose the concept of *heuristic test oracles*, by defining a function that determines whether the outputs are "close enough" to be considered equals. This idea was also used in a later empirical study [75] comparing the effectiveness of three different techniques to test programs without oracles: "niche oracle" (i.e., inputs with known expected outputs), metamorphic testing and assertion checking. The study revealed that metamorphic testing outperforms the other techniques, also when testing non-deterministic programs.

Ding et al. [43] proposed a method named *Self-Checked Metamorphic Testing (SCMT)* combining metamorphic testing and structural testing. SCMT checks the code coverage of source and follow-up test cases during test execution to evaluate the quality of metamorphic relations. It is assumed that the higher the coverage, the more effective the metamorphic relation. The test coverage data obtained may be used to refine test cases by creating, replacing or updating metamorphic relations and their test data. It is also suggested that unexpected coverage outcomes could help detect false-positive results, which they define as a metamorphic relation that holds despite the program being faulty. The approach was evaluated using a cellular image processing program with one seeded bug.

Zhu [82] presented JFuzz, a Java unit testing tool using metamorphic testing. In JFuzz, tests are specified in three parts, namely i) source test case inputs (x), ii) possible transformations on the test inputs ($y = \pi - x$), and iii) metamorphic relations implemented as code assertions ($\sin(x) = \sin(\pi - x)$). Once these elements are defined, the tool automatically generates follow-up test cases by applying the transformations to the source test inputs, it executes source and follow-up test cases, and checks whether the metamorphic relations are violated.

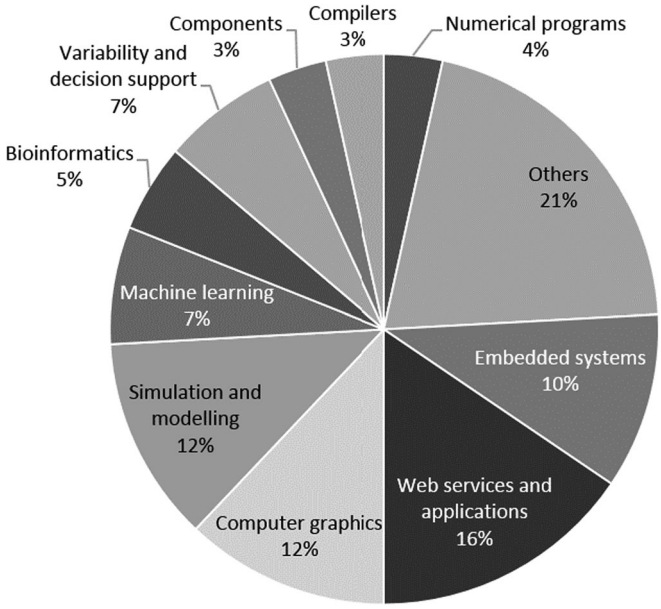


Fig. 4. Metamorphic testing application domains.

5 The Application of Metamorphic Testing

In this section, we answer RQ2 by investigating the scope of metamorphic testing and its applications. In particular, we review applications of metamorphic testing to specific problem domains, and summarise approaches that use metamorphic testing to enhance other testing techniques.

5.1 Application Domains

In this section, we review those papers where the main contribution is a case study on the application of metamorphic testing to specific testing problems (58 out of 119). Fig. 4 classifies these papers according to their application domain. In total, we identified more than 12 different application areas. The most popular domains are web services and applications (16 percent) followed by computer graphics (12 percent), simulation and modeling (12 percent) and embedded systems (10 percent). We also found a variety of applications to other fields (21 percent) such as financial software, optimisation programs or encryption programs. Each of the other domains is explored in no more than four papers, to date. Interestingly, we found that only 4 percent of the papers reported results in numerical programs, even though this seems to be the dominant domain used to illustrate metamorphic testing in the literature.

Fig. 5 shows the domains where metamorphic testing applications have been reported in chronological order. Domains marked with (T) were only explored theoretically. As illustrated, the first application of metamorphic testing was reported in the domain of numerical programs back in 2002. While in the subsequent years the potential applications of metamorphic testing were mainly explored at a theoretical level, there are applications in multiple domains from 2007 onwards. The rest of this section introduces the papers reporting results in each application domain.

5.1.1 Web Services and Applications

Chan et al. [83], [84] presented a metamorphic testing methodology for Service-Oriented Applications (SOA). Their method relies on the use of so-called *metamorphic services* to encapsulate the services under test, execute source and follow-up test cases and check their results. Similarly, Sun et al. [34], [85] proposed to manually derive metamorphic relations from the WSDL description of web services. Their technique automatically generates random source test cases from the WSDL specification and applies the metamorphic relations. They presented a tool to partially automate the process, and evaluated it with three subject web services and mutation analysis. In a related project, Castro-Cabrera and Medina-Bulo [86], [87] presented a metamorphic testing-based approach for web service compositions using the Web Service Business Process Execution Language (WS-BPEL) [88]. To this end, they proposed to analyse the XML description of the service composition to select adequate metamorphic relations. Test cases were defined in terms of the inputs and outputs of the participant services.

In a related set of papers, Zhou et al. [20], [89] used metamorphic testing for the detection of inconsistencies in online web search applications. Several metamorphic relations were proposed and used in a number of experiments with the web search engines Google, Yahoo! and Live Search. Their results showed that metamorphic testing effectively detected inconsistencies in the searches in terms of both returned content and ranking quality. In later work [90], the authors performed an extensive empirical study on the web

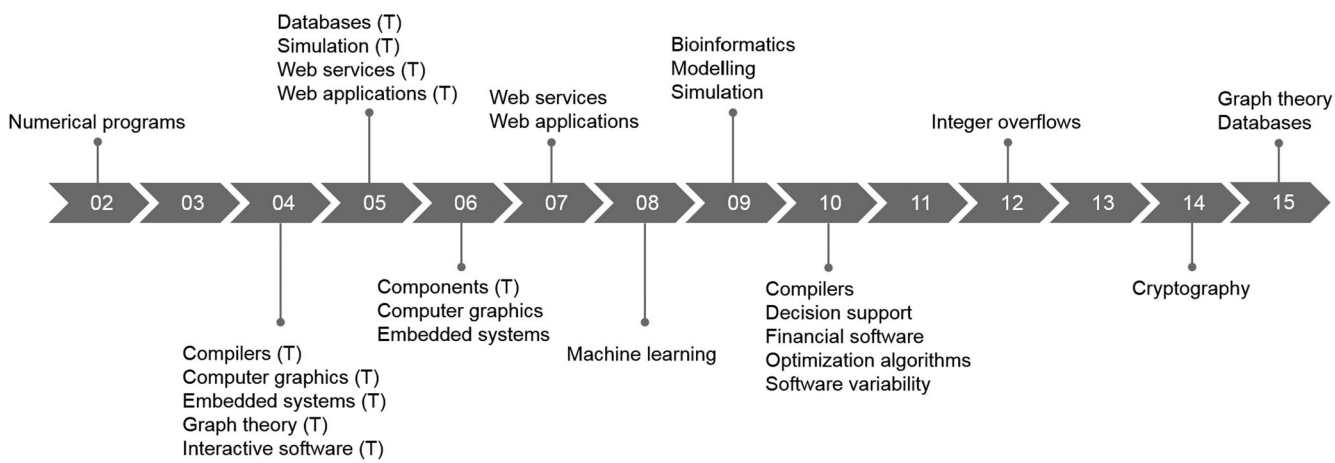


Fig. 5. Timeline of metamorphic testing applications. Domains marked with (T) were only explored theoretically.

search engines Google, Bing, Chinese Bing and Baidu. As a novel contribution, metamorphic relations were defined from the user perspective, representing the properties that a user expects from a "good" search engine, regardless of how the engine is designed. In practice, as previously noticed by Xie et al. [31], this means that metamorphic relations are not only suitable to detect faults in the software under test (verification) but also to check whether the program behaves as the user expects (validation). The authors also proposed using metamorphic testing to assess quality related properties such as reliability, usability or performance. Experimental results revealed a number of failures in the search engines under test.

5.1.2 Computer Graphics

Mayer and Guderlei [91], [92] compared several random image generation techniques for testing image processing programs. The study was performed on the implementation of several image operators as the Euclidean distance transform. Several metamorphic relations were used for the generation of follow-up test cases and the assessment of test results. Chan et al. [93], [94] presented a testing approach for mesh simplification programs using pattern classification and metamorphic testing. Metamorphic relations were used to detect test cases erroneously labelled as passed by a trained pattern classifier. Just and Schweiggert [95] used mutation analysis to evaluate the effectiveness of test data generation techniques and metamorphic relations for a jpeg2000 image encoder. Kuo et al. [33] presented a metamorphic testing approach for programs dealing with the surface visibility problem. A real bug was revealed in a binary space partitioning tree program. Finally, Jameel et al. [96] presented a case study on the application of metamorphic testing to detect faults in morphological image operations such as dilation and erosion. Eight metamorphic relations were reported and assessed on the detection of seeded faults in a binary image dilation program.

5.1.3 Embedded Systems

Tse et al. [97] proposed the application of metamorphic testing to context-sensitive middleware-based software programs. Context-based applications adapt their behaviour according to the information from its environment referred to as *context*. The process of updating the context information typically relies on a *middleware*. Intuitively, their approach generates different context situations and checks whether the outcomes of the programs under test satisfy certain relations. This work was extended to deal with changes in the context during test execution [52], [98]. Chan et al. [99] applied metamorphic testing to wireless sensor networks. As a novel contribution, they proposed to check not only the functional output of source and follow-up test cases but also the energy consumed during the execution, thus targeting both functional and non-functional bugs. Kuo et al. [100] reported a case study on the use of metamorphic testing for the detection of faults in a wireless metering system. A metamorphic relation was identified and used to test the meter reading function of a commercial device from the electric industry in which two real defects were uncovered. Finally, Jiang et al. [101] presented several metamorphic relations for the detection of faults in Central Processing Unit (CPU) scheduling algorithms. Two real bugs were detected in one of the simulators under test.

5.1.4 Simulation and Modelling

Sim et al. [102] presented an application of metamorphic testing for casting simulation, exploiting the properties of the medial axis geometry function. Several metamorphic relations were introduced but no empirical results were presented. Chen et al. [103] proposed the application of metamorphic testing to check the conformance between network protocols and network simulators. A case study was presented testing the OMNeT++ simulator [104] for conformance with the ad-hoc on-demand distance vector protocol. In a related project, Chen et al. [37] proposed using metamorphic testing for the detection of faults in open queuing network modelling, a technique for planning the capacity of computer and communication systems. Ding et al. [105] presented a case study on the detection of faults in a Monte Carlo modelling program for the simulation of photon propagation. Based on their previous work [43], the authors used code coverage criteria to guide the selection of effective metamorphic relations and the creation of test cases. Murphy et al. [76] proposed using metamorphic relations to systematically test health care simulation programs, and presented a case study with two real-world simulators and mutation testing. More recently, Núñez and Hierons [106] proposed using metamorphic relations to detect unexpected behaviour when simulating cloud provisioning and usage. A case study using two cloud models on the iCanCloud simulator [107] was reported. Cañizares et al. [108] presented some preliminary ideas on the use of simulation and metamorphic testing for the detection of bugs related to energy consumption in distributed systems as cloud environments.

5.1.5 Machine Learning

Murphy et al. [58] identified six metamorphic relations that they believe exist in most machine learning applications, namely: additive, multiplicative, permutative, invertive, inclusive, and exclusive relations. The effectiveness of the relations was assessed on three specific machine learning tools in which some real bugs were detected. In a related project, Xie et al. [31], [109] proposed using metamorphic testing for the detection of faults in supervised classifiers. It was argued that metamorphic relations may represent both necessary and expected properties of the algorithm under test. Violations of necessary properties are caused by faults in the algorithm and therefore are helpful for the purpose of verification. Violations of expected properties indicate divergences between what the algorithm does and what the user expects, and thus are helpful for the purpose of validation. Two specific algorithms were studied: K-Nearest neighbors and Naïve Bayes classifier. The results revealed that both algorithms violated some of the necessary properties identified as metamorphic relations indicating faults or unexpected behaviours. Also, some real faults were detected in the open-source machine learning tool Weka [110]. Finally, Jing et al. [111] presented a set of metamorphic relations for association rule algorithms and evaluated them using a contact-lenses data set and the Weka tool.

5.1.6 Variability and Decision Support

Segura et al. [69], [72] presented a test data generator for feature model analysis tools. Test cases are automatically generated from scratch using step-wise transformations that ensure that certain constraints (metamorphic relations) hold at each step. In later work [73], the authors generalised their approach to other variability domains, namely CUDF documents and Boolean formulas. An extensive evaluation of effectiveness showed, among other results, fully automatic detection of 19 real bugs in 7 tools. In a related domain,[5] Kuo et al. [45] presented a metamorphic testing approach for the automated detection of faults in decision support systems. In particular, they focused on the so-called multi-criteria group decision making, in which decision problems are modelled as a three-dimensional matrix representing alternatives, criteria and experts. Several metamorphic relations were presented and used to test the research tool Decider [45], where a bug was uncovered.

5.1.7 Bioinformatics

Chen et al. [40] presented several metamorphic relations for the detection of faults in two open-source bioinformatics programs for gene regulatory networks simulations and short sequence mapping. Also, the authors discussed how metamorphic testing could be used to address the oracle problem in other bioinformatics domains such as phylogenetic, microarray analysis and biological database retrieval. Pullum and Ozmen [112] proposed using metamorphic testing for the detection of faults in predictive models for disease spread. A case study on the detection of faults in two disease–spread models of the 1918 Spanish flu was presented, revealing no bugs. In a related project, Ramanathan et al. [113] proposed using metamorphic testing, data visualisation, and model checking techniques to formally verify and validate compartmental epidemiological models.

5.1.8 Components

Beydeda [114] proposed a self-testing method for commercial off-the-shelf components using metamorphic testing. In this method, components are augmented with self-testing functionality including test case generation, execution and evaluation. In practice, this method allows users of a component to test it even without access to its source code. Lu et al. [115] presented a metamorphic testing methodology for component–based software applications, both at the unit and integration level. The underlying idea is to run test cases against the interfaces of the components under test, using metamorphic relations to construct follow-up test cases and to check their results.

5.1.9 Numerical programs

Chen et al. [116] presented a case study on the application of metamorphic testing to programs implementing partial differential equations. The case study focused on a practical problem in thermodynamics, namely the distribution of temperatures in a square plate. They injected a seeded fault in the program under test and compared the effectiveness of "special" test cases and metamorphic testing in detecting the fault. Special test cases were unable to detect the fault, while metamorphic testing was effective at revealing it using a single metamorphic relation. Aruna and Prasad [117] presented several metamorphic relations for multiplication and division of multi-precision arithmetic software applications. The work was evaluated with four real-time mathematical projects and mutation analysis.

5.1.10 Compilers

Tao et al. [118] presented a so-called "equivalence preservation" metamorphic relation to test compilers. Given an input program, the relation is used to generate an equivalent variant of it, checking whether the behaviours of the resulting executables are the same for a random set of inputs. The authors proposed three different strategies for the generation of equivalent source programs, such as replacing an expression with an equivalent one (e.g., $e \times 2 \equiv e + e$). The evaluation of their approach revealed two real bugs in two C compilers. A closely related idea was presented by Le et al. [119]. Given a program and a set of input values, the authors proposed to create equivalent versions of the program by profiling its execution and pruning unexecuted code. Once a program and its equivalent variant are constructed, both are used as input of the compiler under test, checking for inconsistencies in their results. So far, this method has been used to detect 147 confirmed bugs in two open source C compilers, GCC and LLVM.

5.1.11 Other Domains

Zhou et al. [39] presented several illustrative applications of metamorphic testing in the context of numerical programs, graph theory, computer graphics, compilers and interactive software. Chen et al. [120] claimed that metamorphic testing is both practical and effective for end-user programmers. To support their claim, the authors briefly suggested how metamorphic relations could be used to detect bugs in spreadsheet, database and web applications. Sim et al. [121] presented a metamorphic testing approach for financial software. Several metamorphic relations were integrated into the commercial tool MetaTrader [122] following a self-testing strategy. Source and follow-up test cases were derived from the real-time input price data received at different time periods. Metamorphic testing has also been applied to optimisation programs using both stochastic [123] and heuristic algorithms [32]. Yao et al. [124], [125], [126] presented preliminary results on the use of metamorphic testing to detect integer overflows. Batra and Singh [127] proposed using UML diagrams to guide the selection of metamorphic relations and presented a small case study using a banking application. Sun et al. [128] reported several metamorphic relations for encryption programs. Aruna and Prasad [129] presented a small case study on the application of metamorphic testing to two popular graph theory algorithms. Finally, Lindvall et al. [130] presented an experience report on the use of metamorphic testing to address acceptance testing of NASA's Data Access Toolkit (DAT). DAT is a huge database of telemetry data collected from different NASA missions, and an advance

5. Note that variability models can be used as decision models during software configuration.

query interface to search and mine the available data. Due to the massive amount of data contained in the database, checking the correctness of the query results is challenging. To address this issue, metamorphic testing was used by formulating the same query in different equivalent ways, and asserting that the resulting datasets are the same. Several issues were detected with this approach.

5.2 Other Testing Applications

Besides direct application as a testing technique, metamorphic testing has been integrated into other testing techniques, in order to improve their applicability and effectiveness. In this section, we review these approaches.

Chen et al. [18], [131] proposed using metamorphic testing with fault-based testing. Fault-based testing uses symbolic evaluation [132], [133] and constraint solving [133] techniques to prove the absence of certain types of faults in the program under test. The authors used several numerical programs to illustrate how real and symbolic inputs can be used to discard certain types of faults even in the absence of an oracle. In a related project [30], [134], the authors presented a method called *semi-proving* integrating global symbolic execution and constraint solving for program proving, testing and debugging. Their method uses symbolic execution to prove whether the program satisfies certain metamorphic relations or identify the inputs that violate them. It also supports debugging by identifying violated constraint expressions that reveal failures.

Dong et al. [135] proposed improving the efficiency of Structural Evolutionary Testing (SET) using metamorphic relations. In SET, evolutionary algorithms are used to generate test data that satisfy a certain coverage criteria (e.g., condition coverage). This is often achieved by minimising the distance of the test input to execute the program conditions in the desired way. To improve the efficiency of the process, the authors proposed to use metamorphic relations during the search to consider both source and follow-up test cases as candidate solutions, accelerating the chances of reaching the coverage target. Their approach was evaluated with two numerical programs.

Xie et al. [136], [137] proposed the combination of metamorphic testing and Spectrum-Based Fault Localisation (SBFL) for debugging programs without an oracle. SBFL uses the results of test cases and the corresponding coverage information to estimate the risk of each program entity (e.g., statements) of being faulty. Rather than a regular test oracle, the authors proposed to use the violation or non-violation information from metamorphic relations rather than the actual output of test cases. Among other results, their approach was used to uncover two real bugs in the Siemens Suite [138]. In a related project, Lei et al. [139] applied the same idea to address the oracle problem in a variant of SBFL named Backward-Slice Statistical Fault Localisation (BSSFL) [140]. Rao et al. [141] investigated the ratio between non-violated and violated metamorphic relations in SBFL. They concluded that the higher the ratio of non-violated metamorphic relations to violated metamorphic relations, the less effective the technique. Aruna et al. [142] proposed integrating metamorphic testing with the Ochiai algorithm [143] for fault localisation in dynamic web applications. Five metamorphic relations for a classification algorithm were presented as well as some experimental results.

Liu et al. [144] presented a theoretical description of a new method called *Metamorphic Fault Tolerance (MFT)*. In MFT, the trustworthiness of test inputs is determined in terms of the number of violated and non-violated metamorphic relations. The more relations are satisfied and the fewer relations are violated, the more trustworthy the input is. Also, if an output is judged as untrustworthy, the outputs provided by metamorphic relations can be used to provide a more accurate output.

Jin et al. [145] presented an approach called *Concolic Metamorphic Debugging*, which integrates concolic testing, metamorphic testing, and branch switching debugging, in order to localise potential bugs. Concolic testing is a technique that executes the program under test with both, symbolic and concrete inputs, and then uses symbolic path conditions to derive new test inputs for paths not yet explored. Based on a failure-inducing test input, the proposed method explores all possible program paths in depth-first-order, searching for the first one that passes the metamorphic relation. The final goal is to isolate a minimum amount of code to obtain a passing input, and use that isolation point to localise the fault. The approach, implemented in a tool called Comedy, was evaluated on 21 small programs with seeded faults. Comedy successfully generated debugging reports in 88 percent of the faulty programs and precisely located the fault in 36 percent of them.

6 Experimental Evaluations

In this section, we address RQ3 by reviewing the experimental evaluations of the surveyed papers. In particular, we summarise their main characteristics in terms of subject programs, source test cases, types of faults, number of metamorphic relations and evaluation metrics. Additionally, we review the results of empirical studies involving humans.

6.1 Subject Programs

As a part of the review process, we collected information about the subject programs used for the evaluation of metamorphic testing contributions. The table provided as supplemental material, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TSE.2016.2532875, shows the name, language, size, description and the references of the papers reporting results for each program. In the cases where the information was unavailable in the literature, it is indicated with "NR" (Not Reported). The table is ordered by the number of papers that use the subject programs. Thus, the programs at the top of the list are the most studied subject programs in the metamorphic testing literature. Overall, we identified 145 different subject programs. Most of them are written in Java (46.2 percent) and C/C++ (35.5 percent), with reported sizes ranging between 12 and 12,795 lines of code.

In experimentation, the use of real world programs, rather than research programs, is commonly recognised as an indicator of the maturity of a discipline [25]. To assess this maturity, we studied the relationship between the use of research and real world programs in metamorphic testing experiments. Similarly to previous surveys [25], we consider a program to

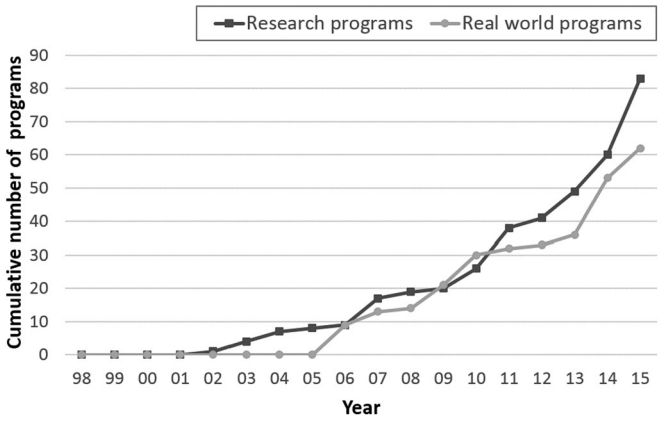


Fig. 6. Research versus real world subject programs.

be a "real world" program if it is either a commercial or an open-source program, otherwise we consider it as a "research program". As an exception to this rule, we consider all open source projects that are designed as benchmarks rather than applications as research programs (e.g., the Siemens suite). Fig. 6 presents the cumulative view of the number of each type of program, research and real world, by year. As illustrated, research programs are used since 2002, while real world programs were not introduced in metamorphic testing experiments until 2006. Since then, the use of both types of programs has increased with similar trends. It is noteworthy that the number of real world programs in 2010 was higher than the number of research programs. The cumulative number in 2015 shows a significant advantage of research programs (83) over real world programs (62). The overall trend, however, suggests that metamorphic testing is maturing.

6.2 Source Test Cases

Metamorphic testing requires the use of source test cases that serve as seed for the generation of follow-up test cases. Source test cases can be generated using any traditional testing techniques. We studied the different techniques used in the literature and counted the number of papers using each of them; the results are presented in Fig. 7. As illustrated, a majority of studies used random testing for the generation of source test cases (57 percent), followed by those using an existing test suite (34 percent). Also, several papers (6 percent) use tool-based techniques such as constraint programming, search-based testing or symbolic execution. This diversity of usable sources supports the applicability of metamorphic testing. It also supports the use of random testing as a cost-effective and straightforward approach for the generation of the initial test suite (cf., Section 4.3).

6.3 Types of Faults

The effectiveness of metamorphic testing approaches is assessed according to their ability to detect failures caused by faults in the programs under test. Uncovering real bugs is the primary goal, but they are not always available for evaluation. Thus, most authors introduce artificial faults (a.k.a. mutants) in the subject programs either manually or automatically, using mutation testing tools [25]. To study the relationship between real bugs and mutants in metamorphic testing evaluations, we calculated the cumulative number of

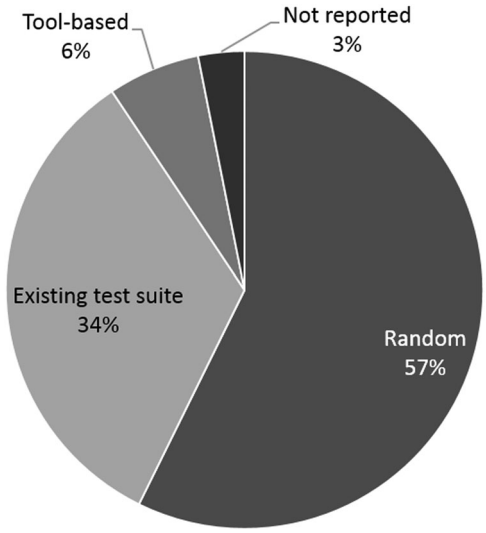


Fig. 7. Source test case generation techniques.

papers reporting results with artificial and real bugs by year, depicted in Fig. 8. We consider a real bug to be a latent, initially unknown, fault in the subject program. As illustrated in Fig. 8, the first experimental results with mutants were presented back in 2002, while the first real bugs were reported in 2007. Since then, the number of papers reporting results with both types of faults has increased, although artificial faults show a steeper angle representing a stronger trend. Besides this, we also counted the number of faults used in each paper. To date, metamorphic testing has been used to detect about 295 distinct real faults in 36 different tools, 23 of which are real world programs, suggesting that metamorphic testing is effective at detecting real bugs.

6.4 Metamorphic Relations

The number of metamorphic relations used in experimentation may be a good indicator of the effort required to apply metamorphic testing. As a part of the data collection process, we counted the number of metamorphic relations presented in each paper containing experimental results. Fig. 9 classifies the papers based on the number of metamorphic relations reported. As illustrated, the largest portion of studies report between five and nine metamorphic relations (39 percent), followed by those presenting between one and four metamorphic relations (24 percent). Interestingly, only nine studies (13 percent) presented more than 25 metamorphic

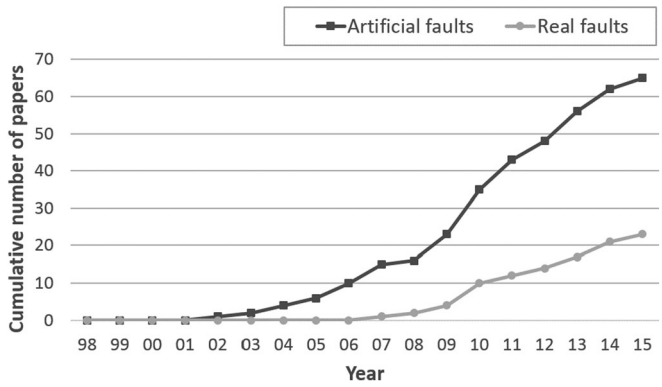


Fig. 8. Artificial versus real faults.

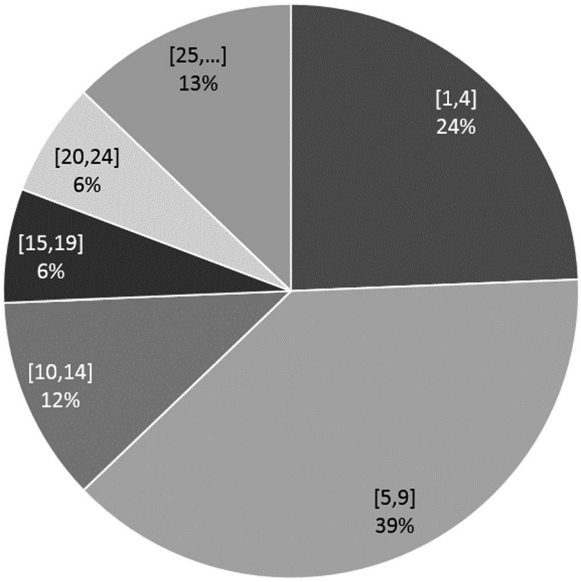


Fig. 9. Number of metamorphic relations.

relations. We took a closer look at those nine papers and observed that all of them reported results for several subject programs. These findings suggest that a modest number of metamorphic relations (less than 10) is usually sufficient to apply metamorphic testing with positive results.

6.5 Evaluation Metrics

Numerous metrics to evaluate the effectiveness of metamorphic testing approaches have been proposed. Among them, we identified two metrics intensively used in the surveyed papers, such that they could be considered as a de-facto standard in the metamorphic testing literature.

6.5.1 Mutation Score

This metric is based on mutation analysis, where mutation operators are applied to systematically produce versions of the program under test containing artificial faults ("mutants") [25]. The mutation score is the ratio of detected ("killed") mutants to the total number of mutants. Mutants that do not change the program's semantics and thus cannot be detected are referred to as *equivalent* [25]. In theory, equivalent mutants should be excluded from the total number of mutants, but in practice this is not always possible since program equivalence is undecidable. Suppose a metamorphic test suite t composed of a set of metamorphic tests, i.e., pairs of source and follow-up test cases. The Mutation Score (MS) of t is calculated as follows:

$$MS(t) = \frac{M_k}{M_t - M_e}, \tag{1}$$

where M_k is the number of killed mutants by the metamorphic tests in t, M_t is the total number of mutants and M_e is the number of equivalent mutants. A variant of this metric [71], [91], [121] is often used to calculate the ratio of mutants detected by a given metamorphic relation r as follows:

$$MS(t, r) = \frac{M_{kr}}{M_t - M_e}, \tag{2}$$

where M_{kr} is the number of mutants killed by the metamorphic tests in t derived from r. This metric is also called mutation detection ratio [36].

6.5.2 Fault Detection Ratio

This metric calculates the ratio of test cases that detect a given fault [41], [55], [68], [70], [71], [101], [124], [126]. The Fault Detection Ratio (FDR) of a metamorphic test suite t and a fault f is calculated as follows:

$$FDR(t, f) = \frac{T_f}{T_t}, \tag{3}$$

where T_f is the number of tests that detect f and T_t is the total number of tests in t. A variant of this metric [27], [32], [33], [37], [45], [54], [71] calculates the ratio of test cases that detect a fault f using a given metamorphic relation r as follows:

$$FDR(t, f, r) = \frac{T_{fr}}{T_r}, \tag{4}$$

where M_{fr} is the number of tests in t derived from the relation r that detect the fault f, and T_r is the total number of metamorphic tests derived from r. This metric is also called fault discovery rate [34], [85], [128].

6.6 Empirical Studies with Humans

Hu et al. [29], [36] reported on a controlled experiment to investigate the cost-effectiveness of using metamorphic testing by 38 testers on three open-source programs. The experiment participants were either asked to write metamorphic relations, or tests with assertions to check whether the final or intermediate state of the program under test is correct. The experiment revealed a trade-off between both techniques, with metamorphic testing being less efficient but more effective at detecting faults than tests with assertions.

Liu et al. [146] reported on a three-year experience in teaching metamorphic testing to various groups of students at Swinburne University of Technology (Australia). The authors explained the teaching approach followed and the lesson learned, concluding that metamorphic testing is a suitable technique for end-user testing. In a later paper, Liu et al. [4] presented an empirical study to investigate the effectiveness of metamorphic testing addressing the oracle problem compared with random testing. For the study, several groups of undergraduate and postgraduate students from two different universities were recruited to identify metamorphic relations in five subject programs of algorithmic type. Metamorphic testing was compared to random testing with and without oracle. Their experiment showed that metamorphic testing was able to find more faults than random testing with and without oracle in most subject programs. Furthermore, it was concluded that a small number of diverse metamorphic relations (between 3 and 6), even those identified in an ad-hoc manner, had a similar fault-detection capability to a test oracle, i.e., comparing the program output with the expected one.

7 Challenges

A number of open research challenges emerge from this survey, based on problems repeatedly encountered throughout

the reviewed papers, or gaps in the literature. These challenges answer RQ4.

Challenge 1. Guidelines for the construction of good metamorphic relations. For most problems, a variety of metamorphic relations with different fault-detection capability can be identified. It is therefore key to know the properties of effective metamorphic relations and to provide systematic methods for their construction. Although several authors have reported lessons learned on the properties of good metamorphic relations (cf., Section 4.1), these are often complementary or even contradictory (e.g., [27], [38]). Therefore, there is a lack of reliable guidelines for the construction of effective metamorphic relations. Such guidelines should provide a step-by-step process to guide testers, both experts and beginners, in the construction of good metamorphic relations.

Challenge 2. Prioritisation and minimisation of metamorphic relations. In certain cases using all the available metamorphic relations may be too expensive and a subset of them must be selected. It is therefore important to know how to prioritise the most effective metamorphic relations. To this end, several authors have proposed using code coverage [43], [46] or test case similarity [47] with promising results. However, the applicability of those approaches as domain-independent prioritisation criteria still needs to be explored. Furthermore, analogously to the concept of test suite minimisation, where redundant test cases are removed from a suite as it evolves [147], the use of minimisation techniques to remove redundant metamorphic relations is an open problem where research is needed. It is worth mentioning that test case minimisation is a NP-hard problem and therefore heuristic techniques should be explored.

Challenge 3. Generation of likely metamorphic relations. The generation of metamorphic relations is probably the most challenging problem to be addressed. Although some promising results have been reported, those are mainly restricted to the scope of numerical programs. The generation of metamorphic relations in other domains as well as the use of different techniques for rule inference are topics where contributions are expected. We also foresee a fruitful line of research exploring the synergies between the problem of generating metamorphic relations and the detection of program invariants [64], [148].

Challenge 4. Combination of metamorphic relations. As presented in Section 4.2, several authors have explored the benefits of combining metamorphic relations following two different strategies, namely applying metamorphic relations in a chain style (IMT) and composing metamorphic relations to construct new relations (CMR). It remains an open problem, however, to compare both approaches and to provide heuristics to decide when to use one or the other. Also, these techniques raise new research problems such us determining whether a given set of metamorphic relations can be combined and in which order.

Challenge 5. Automated generation of source test cases. As described in Section 4.3, most papers use either randomly generated or existing test suites as source tests when applying metamorphic testing. However, there is evidence that the source test cases influence the effectiveness of metamorphic relations [28], [68], [69]. Promising initial results in generating source test cases specifically for given metamorphic relations have been achieved, but many open questions remain about what constitutes the best possible source test cases and how to generate them.

Challenge 6. Metamorphic testing tools. Only two out of all 119 presented a tool as main contribution [78], [82], and very few of the papers on metamorphic testing mentioned a tool implementing the presented techniques [64], [65], [67], [73], [81], [89], [118], [145]. Indeed, if practitioners want to apply metamorphic testing today, they would have to implement their own tool, as there are no publicly available and maintained tools. This is a significant obstacle for a wide-spread use of metamorphic testing in empirical research as well as in practice.

8 Conclusions

In this article, we presented a survey on metamorphic testing covering 119 papers published between 1998 and 2015. We analysed ratios and trends indicating the main advances on the technique, its application domains and the characteristics of experimental evaluations. The results of the survey show that metamorphic testing is a thriving topic with an increasing trend of contributions on the subject. We also found evidence of the applicability of the technique to multiple domains far beyond numerical programs, as well as its integration with other testing techniques. Furthermore, we identified an increasing number of papers reporting the detection of faults in real world programs. All these findings suggest that metamorphic testing is gaining maturity as an effective testing technique, not only to alleviate the oracle problem, but also for the automated generation of test data. Finally, despite the advances on metamorphic testing, our survey points to areas where research is needed. We trust that this work may become a helpful reference for future development on metamorphic testing as well as to introduce newcomers in this promising testing technique.

Acknowledgments

We would like to thank T. Y. Chen, Robert M. Hierons, Phil McMinn, Amador Durán, Zhi Quan Zhou, Christian Murphy, Huai Liu, Xiaoyuan Xie, Alberto Goffi, Gagandeep, Carmen Castro-Cabrera, Yan Lei and Peng Wu for their helpful comments in an earlier version of this article. We are also grateful to the members of the SSE research group led by Mark Harman for the insightful and inspiring discussion during our visit at the University College London. This work has been partially supported by the European Commission (FEDER) and Spanish Government under CICYT projects TAPAS (TIN2012-32273) and BELI (TIN2015-70560-R), and the Andalusian Government projects THEOS (TIC-5906) and COPAS (P12-TIC-1867).

References

[1] E. J. Weyuker, "On testing non-testable programs," *Comput. J.*, vol. 25, no. 4, pp. 465–470, 1982.

[2] S. Anand, E. K. Burke, T. Y. Chen, J. Clark, M. B. Cohen, W. Grieskamp, M. Harman, M. J. Harrold, and P. Mcminn. (Aug. 2013). An orchestrated survey of methodologies for automated software test case generation. *J. Syst. Softw.*[Online] *86(8)*, pp. 1978–2001. Available: http://dx.doi.org/10.1016/j.jss.2013.02.061.

[3] E. T. Barr, M. Harman, P. McMinn, M. Shahbaz, and S. Yoo, "The oracle problem in software testing: A survey," *IEEE Trans. Softw. Eng.*, vol. 41, no. 5, pp. 507–525, May 2015.

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

[5] T. Y. Chen, S. C. Cheung, and S. M. Yiu, "Metamorphic testing: A new approach for generating next test cases," Dept. Comput. Sci., The Hong Kong Univ. Sci. Technol., Hong Kong, Tech. Rep. HKUST-CS98-01, 1998.

[6] W. J. Cody, *Software Manual for the Elementary Functions*. Upper Saddle River, NJ, USA: Prentice-Hall, 1980.

[7] M. Blum, M. Luby, and R. Rubinfeld. (1993). Self-testing/correcting with applications to numerical problems. *J. Comput. Syst. Sci.* [Online]. *47(3)*, pp. 549–595. Available: http://www.sciencedirect.com/science/article/pii/002200009390044W.

[8] P. E. Ammann and J. C. Knight. (Apr. 1988). Data diversity: An approach to software fault tolerance. *IEEE Trans. Comput.* [Online]. *37(4)*, pp. 418–425. Available: http://dx.doi.org/10.1109/12.2185.

[9] T. Y. Chen, F.-C. Kuo, T. H. Tse, and Z. Q. Zhou, "Metamorphic testing and beyond," in *Proc. 11th Annu. Int. Workshop Softw. Technol. Eng. Practice*, Sep. 2003, pp. 94–100.

[10] T. H. Tse, "Research directions on model-based metamorphic testing and verification," in *Proc. 29th Annu. Int. Comput. Softw. Appl. Conf.*, vol. 1, Jul. 2005, p. 332.

[11] T. Y. Chen, "Metamorphic testing: A simple approach to alleviate the oracle problem," in *Proc. 5th IEEE Int. Symp. Service Oriented Syst. Eng.*, Jun. 2010, pp. 1–2.

[12] W. K. Chan and T. H. Tse, "Oracles are hardly attain'd, and hardly understood: Confessions of software testing researchers," in *Proc. 13th Int. Conf. Quality Softw.*, Jul. 2013, pp. 245–252.

[13] T. Y. Chen, "Metamorphic testing: A simple method for alleviating the test oracle problem," in *Proc. 10th Int. Workshop Autom. Softw. Test*, 2015, pp. 53–54. [Online]. Available: http://dl.acm.org/citation.cfm?id=2819261.2819278.

[14] T. Y. Chen, F.-C. Kuo, D. Towey, and Z. Q. Zhou, "Metamorphic testing: Applications and integration with other methods: Tutorial synopsis," in *Proc. 12th Int. Conf. Quality Softw.*, Aug. 2012, pp. 285–288.

[15] Z. Hui and S. Huang, "Achievements and challenges of metamorphic testing," in *Proc. 4th World Congr. Softw. Eng.*, Dec. 2013, pp. 73–77.

[16] U. Kanewala and J. M. Bieman, "Techniques for testing scientific programs without an oracle," in *Proc. 5th Int. Workshop Softw. Eng. Comput. Sci. Eng.*, May 2013, pp. 48–57.

[17] G. Dong, B. Xu, L. Chen, C. Nie, and L. Wang, "Survey of metamorphic testing," *J. Frontiers Comput. Sci. Technol.*, vol. 3, no. 2, pp. 130–143, 2009.

[18] T. Y. Chen, T. H. Tse, and Z. Q. Zhou. (2003). Fault-based testing without the need of oracles. *Inform. Softw. Technol.* [Online]. *45 (1)*, pp. 1–9. Available: http://dx.doi.org/10.1016/S0950-5849(02)00129-5.

[19] M. D. Ernst, J. H. Perkins, P. J. Guo, S. McCamant, C. Pacheco, M. S. Tschantz, and C. Xiao. (2007, Dec.). The Daikon system for dynamic detection of likely invariants. *Sci. Comput. Program.* [Online]. *69(1-3)*, pp. 35–45. Available: http://dx.doi.org/10.1016/j.scico.2007.01.015.

[20] Z. Q. Zhou, S. Zhang, M. Hagenbuchner, T. H. Tse, F.-C. Kuo, and T. Y. Chen. (Jun. 2012). Automated functional testing of online search services. *Softw. Testing, Verification Rel. J.* [Online]. *22(4)*, pp. 221–243. Available: http://dx.doi.org/10.1002/stvr.437.

[21] B. Kitchenham, "Procedures for performing systematic reviews," Keele Univ. and NICTA, Staffordshire, U.K., Tech. Rep. TR/SE-0401, 2004.

[22] J. Webster and R. Watson. (2002). Analyzing the past to prepare for the future: Writing a literature review. *MIS Quart.* [Online]. *26(2)*, pp. xiii–xxiii. Available: http://www.misq.org/archivist/vol/no26/issue2/GuestEd.pdf.

[23] D. Benavides, S. Segura, and A. Ruiz-Cortés, "Automated analysis of feature models 20 years later: A literature review," *Inform. Syst.*, vol. 35, no. 6, pp. 615–636, 2010.

[24] Y. J. M. Harman and Y. Zhang, "Achievements, open problems and challenges for search based software testing," in *Proc. 8th IEEE Int. Conf. Softw. Testing, Verification Validation*, Apr. 2015, pp. 1–12.

[25] Y. Jia and M. Harman. (2011, Sep.). An analysis and survey of the development of mutation testing. *IEEE Trans. Softw. Eng.*, vol. 37, no. 5, pp. 649–678, Sep. 2011. [Online]. Available: http://dx.doi.org/10.1109/TSE.2010.62.

[26] S. Segura, G. Fraser, A. B. Sánchez, and A. Ruiz-Cortés. (2016, Feb.). Metamorphic testing: A literature review. Appl. Softw. Eng. Res. Group, Univ. Seville, Seville, Spain. Tech. Rep. ISA-16-TR-02. [Online]. Available: http://www.isa.us.es/sites/default/files/isa-16-tr-02_0.pdf.

[27] T. Y. Chen, D. H. Huang, T. H. Tse, and Z. Q. Zhou, "Case studies on the selection of useful relations in metamorphic testing," in *Proc. 4th Ibero-Amer. Symp. Softw. Eng. Knowl. Eng.*, 2004, pp. 569–583.

[28] T. Y. Chen, F.-C. Kuo, Y. Liu, and A. Tang, "Metamorphic testing and testing with special values," in *Proc. 5th ACIS Int. Conf. Softw. Eng., Artif. Intell., Netw. Parallel/Distrib. Comput.*, 2004, pp. 128–134.

[29] Z. Zhang, W. K. Chan, T. H. Tse, and P. Hu, "Experimental study to compare the use of metamorphic testing and assertion checking," *J. Softw.*, vol. 20, no. 10, pp. 2637–2654, 2009.

[30] T. Y. Chen, T. H. Tse, and Z. Q. Zhou, "Semi-proving: An integrated method for program proving, testing, and debugging," *IEEE Trans. Softw. Eng.*, vol. 37, no. 1, pp. 109–125, Jan. 2011.

[31] X. Xie, J. W. K. Ho, C. Murphy, G. Kaiser, B. Xu, and T. Y. Chen. (2011, Apr.). Testing and validating machine learning classifiers by metamorphic testing. *J. Syst. Softw.* [Online]. *84 (4)*, pp. 544–558. Available: http://dx.doi.org/10.1016/j.jss.2010.11.920.

[32] A. C. Barus, T. Y. Chen, D. Grant, F.-C. Kuo, and M. F. Lau, "Testing of heuristic methods: A case study of greedy algorithm," in *Software Engineering Techniques*, Z. Huzar, R. Koci, B. Meyer, B. Walter, and J. Zendulka, Eds. Berlin, Germany: Springer, 2011, vol. 4980, pp. 246–260. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-22386-0_19.

[33] F.-C. Kuo, S. Liu, and T. Y. Chen, "Testing a binary space partitioning algorithm with metamorphic testing," in *Proc. ACM Symp. Appl. Comput.*, 2011, pp. 1482–1489. [Online]. Available: http://doi.acm.org/10.1145/1982185.1982502.

[34] C. Sun, G. Wang, B. Mu, H. Liu, Z. Wang, and T. Y. Chen, "Metamorphic testing for web services: Framework and a case study," in *Proc. IEEE Int. Conf. Web Services*, Jul. 2011, pp. 283–290.

[35] Z.-W. Hui, S. Huang, H. Li, J.-H. Liu, and L.-P. Rao, "Measurable metrics for qualitative guidelines of metamorphic relation," in *Proc. IEEE 39th Annu. Comput. Softw. Appl. Conf.*, vol. 3, Jul. 2015, pp. 417–422.

[36] P. Hu, Z. Zhang, W. K. Chan, and T. H. Tse, "An empirical comparison between direct and indirect test result checking approaches," in *Proc. 3rd Int. Workshop Softw. Quality Assurance*, 2006, pp. 6–13. [Online]. Available: http://doi.acm.org/10.1145/1188895.1188901.

[37] T. Y. Chen, F.-C. Kuo, R. Merkel, and W. K. Tam, "Testing an open source suite for open queuing network modelling using metamorphic testing technique," in *Proc. IEEE 14th Int. Conf. Eng. Complex Comput. Syst.*, Jun. 2009, pp. 23–29.

[38] J. Mayer and R. Guderlei, "An empirical study on the selection of good metamorphic relations," in *Proc. 30th Annu. Int. Comput. Softw. Appl. Conf.*, vol. 1, Sep. 2006, pp. 475–484.

[39] Z. Q. Zhou, D. H. Huang, T. H. Tse, Z. Yang, H. Huang, and T. Y. Chen, "Metamorphic testing and its applications," in presented at the 8th Int. Symp. Future Software Technology, Xian, China, 2004.

[40] T. Y. Chen, J. W. K. Ho, H. Liu, and X. Xie. (2009). An innovative approach for testing bioinformatics programs using metamorphic testing. *BioMed. Central Bioinform. J.* [Online]. *10(1)*, p. 24. Available: http://www.biomedcentral.com/1471-2105/10/24.

[41] G. Batra and J. Sengupta, "An efficient metamorphic testing technique using genetic algorithm," in *Information Intelligence, Systems, Technology and Management* (ser. Communications in Computer and Information Science), S. Dua, S. Sahni, and D. Goyal, Eds., Berlin, Germany: Springer, 2011, vol. 141, pp. 180–188. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-19423-8_19.

[42] L. Chen, L. Cai, J. Liu, Z. Liu, S. Wei, and P. Liu, "An optimized method for generating cases of metamorphic testing," in *Proc. 6th Int. Conf. New Trends Inform. Sci. Service Sci. Data Mining*, Oct. 2012, pp. 439–443.

[43] J. Ding, T. Wu, J. Q. Lu, and X. Hu, "Self-checked metamorphic testing of an image processing program," in *Proc. 4th Int. Conf. Secure Softw. Integr. Rel. Improvement*, Jun. 2010, pp. 190–197.

[44] G. Dong, T. Guo, and P. Zhang, "Security assurance with program path analysis and metamorphic testing," in *Proc. 4th IEEE Int. Conf. Softw. Eng. Service Sci.*, May 2013, pp. 193–197.
[45] F.-C. Kuo, Z. Q. Zhou, J. Ma, and G. Zhang, "Metamorphic testing of decision support systems: A case study," *IET Softw.*, vol. 4, no. 4, pp. 294–301, Aug. 2010.
[46] M. Asrafi, H. Liu, and F.-C. Kuo, "On testing effectiveness of metamorphic relations: A case study," in *Proc. 5th Int. Conf. Secure Softw. Integr. Rel. Improvement*, Jun. 2011, pp. 147–156.
[47] Y. Cao, Z. Q. Zhou, and T. Y. Chen, "On the correlation between the effectiveness of metamorphic relations and dissimilarities of test case executions," in *Proc. 13th Int. Conf. Quality Softw.*, Jul. 2013, pp. 153–162.
[48] Z. Q. Zhou, "Using coverage information to guide test case selection in adaptive random testing," in *Proc. Comput. Softw. Appl. Conf. Workshops*, Jul. 2010, pp. 208–213.
[49] R. Just and F. Schweiggert, "Automating software tests with partial oracles in integrated environments," in *Proc. 5th Workshop Autom. Softw. Test*, 2010, pp. 91–94. [Online]. Available: http://doi.acm.org/10.1145/1808266.1808280.
[50] R. Just and F. Schweiggert. (2011). Automating unit and integration testing with partial oracles. *Softw. Quality J.* [Online]. *19(4)*, pp. 753–769. Available: http://dx.doi.org/10.1007/s11219-011-9151-x.
[51] X. Xie, J. Tu, T. Y. Chen, and B. Xu, "Bottom-up integration testing with the technique of metamorphic testing," in *Proc. 14th Int. Conf. Quality Softw.*, Oct. 2014, pp. 73–78.
[52] W. K. Chan, T. Y. Chen, H. Lu, T. H. Tse, and S. S. Yau. (2006). Integration testing of context-sensitive middleware-based applications: A metamorphic approach. *Int. J. Softw. Eng. Knowl. Eng.* [Online]. *16(5)*, pp. 677–704. Available: http://dblp.uni-trier.de/db/journals/ijseke/ijseke16.html#ChanCLTY06.
[53] Z. Hui and S. Huang, "A formal model for metamorphic relation decomposition," in *Proc. 4th World Congr. Softw. Eng.*, Dec. 2013, pp. 64–68.
[54] H. Liu, X. Liu, and T. Y. Chen, "A new method for constructing metamorphic relations," in *Proc. 12th Int. Conf. Quality Softw.*, Aug. 2012, pp. 59–68.
[55] G. Dong, B. Xu, L. Chen, C. Nie, and L. Wang, "Case studies on testing with compositional metamorphic relations," *J. Southeast Univ. (English Edition)*, vol. 24, no. 4, pp. 437–443, 2008.
[56] U. Kanewala and J. M. Bieman, "Using machine learning techniques to detect metamorphic relations for programs without test oracles," in *Proc. IEEE 24th Int. Symp. Softw. Rel. Eng.*, Nov. 2013, pp. 1–10.
[57] U. Kanewala, "Techniques for automatic detection of metamorphic relations," in *Proc. IEEE 7th Int. Conf. Softw. Testing, Verification Validation Workshops*, Mar. 2014, pp. 237–238.
[58] C. Murphy, G. Kaiser, and L. Hu, "Properties of machine learning applications for use in metamorphic testing," Dept. Comput. Sci., Columbia Univ., New York NY, USA, Tech. Rep. CUCS-011-08, 2008.
[59] U. Kanewala, J. M. Bieman, and A. Ben-Hur. (2015). Predicting metamorphic relations for testing scientific software: A machine learning approach using graph kernels. *Softw. Testing, Verification Rel.* [Online]. Available: http://dx.doi.org/10.1002/stvr.1594.
[60] J. Zhang, J. Chen, D. Hao, Y. Xiong, B. Xie, L. Zhang, and H. Mei, "Search-based inference of polynomial metamorphic relations," in *Proc. 29th ACM/IEEE Int. Conf. Automat. Softw. Eng.*, 2014, pp. 701–712. [Online]. Available: http://doi.acm.org/10.1145/2642937.2642994.
[61] A. Carzaniga, A. Goffi, A. Gorla, A. Mattavelli, and M. Pezzè, "Cross-checking oracles from intrinsic software redundancy," in *Proc. 36th Int. Conf. Softw. Eng.*, 2014, pp. 931–942. [Online]. Available: http://doi.acm.org/10.1145/2568225.2568287.
[62] A. Goffi, A. Gorla, A. Mattavelli, M. Pezze, and P. Tonella, "Search-based synthesis of equivalent method sequences," in *Proc. 22nd ACM SIGSOFT Int. Symp. Found.Softw. Eng.*, 2014, pp. 366–376. [Online]. Available: http://doi.acm.org/10.1145/2635868.2635888.
[63] A. Goffi, "Automatic generation of cost-effective test oracles," in *Proc. 36th Int. Conf. Softw. Eng.*, 2014, pp. 678–681. [Online]. Available: http://doi.acm.org/10.1145/2591062.2591078.
[64] F. Su, J. Bell, C. Murphy, and G. Kaiser, "Dynamic inference of likely metamorphic properties to support differential testing," in *Proc. 10th Int. Workshop Autom. Softw. Test*, 2015, pp. 55–59. [Online]. Available: http://dl.acm.org/citation.cfm?id=2819261.2819279.
[65] T. Y. Chen, P. Poon, and X. Xie. (2015). METRIC: METamorphic Relation Identification based on the category-choice framework. *J. Syst. Softw.* [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0164121215001624.
[66] T. Y. Chen, P.-L. Poon, S.-F. Tang, and T. H. Tse, "Dessert: a divide-and-conquer methodology for identifying categories, choices, and choice relations for test case generation," *IEEE Trans. Softw. Eng.*, vol. 38, no. 4, pp. 794–809, Jul./Aug. 2012.
[67] A. Gotlieb and B. Botella, "Automated metamorphic testing," in *Proc. 27th Annu. Int. Conf. Comput. Softw. Appl.*, 2003, pp. 34–40. [Online]. Available: http://dl.acm.org/citation.cfm?id=950785.950794.
[68] P. Wu, X. Shi, J. Tang, H. Lin, and T. Y. Chen, "Metamorphic testing and special case testing: A case study," *J. Softw.*, vol. 16, pp. 1210–1220, 2005.
[69] S. Segura, R. M. Hierons, D. Benavides, and A. Ruiz-Cortés. Automated metamorphic testing on the analyses of feature models. *Inform. Softw. Technol.* [Online]. *53(3)*, pp. 245–258. Available: http://www.sciencedirect.com/science/article/pii/S0950584910001904.
[70] P. Wu, "Iterative metamorphic testing," in *Proc. 29th Annu. Int. Comput. Softw. Appl. Conf.*, vol. 1, pp. 19–24, Jul. 2005.
[71] G. Dong, C. Nie, B. Xu, and L. Wang, "An effective iterative metamorphic testing algorithm based on program path analysis," in *Proc. 7th Int. Conf. Quality Softw.*, Oct. 2007, pp. 292–297.
[72] S. Segura, R. M. Hierons, D. Benavides, and A. Ruiz-Cortes, "Automated test data generation on the analyses of feature models: A metamorphic testing approach," in *Proc. 3rd Int. Conf. Softw. Testing, Verification Validation*, Apr. 2010, pp. 35–44.
[73] S. Segura, A. Durán, A. B. Sánchez, D. L. Berre, E. Lonca, and A. Ruiz-Cortés. (2015). Automated metamorphic testing of variability analysis tools. *Softw. Testing, Verification Rel.* [Online]. *25(2)*, pp. 138–163. Available: http://dx.doi.org/10.1002/stvr.1566.
[74] R. Guderlei and J. Mayer, "Statistical metamorphic testing: Testing programs with random output by means of statistical hypothesis tests and metamorphic testing," in *Proc. 7th Int. Conf. Quality Softw.*, Oct. 2007, pp. 404–409.
[75] C. Murphy and G. Kaiser. (2010). Empirical evaluation of approaches to testing applications without test oracles. Columbia Univ. Comput. Sci., New York, NY, Tech. Rep. CUCS-039-10. [Online]. Available: http://hdl.handle.net/10022/AC:P:10525.
[76] C. Murphy, M. S. Raunak, A. King, S. Chen, C. Imbriano, G. Kaiser, I. Lee, O. Sokolsky, L. Clarke, and L. Osterweil, "On effective testing of health care simulation software," in *Proc. 3rd Workshop Softw. Eng. Health Care*, 2011, pp. 40–47. [Online]. Available: http://doi.acm.org/10.1145/1987993.1988003.
[77] C. Murphy, "Using runtime testing to detect defects in applications without test oracles," in *Proc. Found.Softw. Eng. Doctoral Symp.*, 2008, pp. 21–24. [Online]. Available: http://doi.acm.org/10.1145/1496653.1496659.
[78] C. Murphy, K. Shen, and G. Kaiser, "Using JML runtime assertion checking to automate metamorphic testing in applications without test oracles," in *Proc. Second Int. Conf. Softw. Testing Verification Validation*, 2009, pp. 436–445.
[79] (2016). "Java Modeling Language (JML). [Online]. Available: http://www.jmlspecs.org.
[80] C. Murphy, G. Kaiser, L. Hu, and L. Wu, "Properties of machine learning applications for use in metamorphic testing," in *Proc. Int. Conf. Softw. Eng. Knowl. Eng.*, 2008, pp. 867–872.
[81] C. Murphy, K. Shen, and G. Kaiser, "Automatic system testing of programs without test oracles," in *Proc. 18th Int. Symp. Softw. Testing Anal.*, 2009, pp. 189–200. [Online]. Available: http://doi.acm.org/10.1145/1572272.1572295.
[82] H. Zhu, "Jfuzz: A tool for automated Java unit testing based on data mutation and metamorphic testing methods," in *Proc. 2nd Int. Conf. Trustworthy Syst. Appl.*, Jul. 2015, pp. 8–15.
[83] W. K. Chan, S. C. Cheung, and K. R. P. Leung, "Towards a metamorphic testing methodology for service-oriented software applications," in *Proc. 5th Int. Conf. Quality Softw.*, Sep. 2005, pp. 470–476.
[84] W. K. Chan, S. C. Cheung, and K. R. P. H. Leung. (2007). A metamorphic testing approach for online testing of service-oriented software applications. *Int. J. Web Serv. Res.* [Online]. *4(2)*, pp. 61–81. Available: http://dblp.uni-trier.de/db/journals/jwsr/jwsr4.html#ChanCL07.

[85] C. Sun, G. Wang, B. Mu, H. Liu, Z. Wang, and T. Y. Chen. (2012, Jan.). A metamorphic relation-based approach to testing web services without oracles. *Int. J. Web Serv. Res.* [Online]. *9(1)*, pp. 51–73. Available: http://dx.doi.org/10.4018/jwsr.2012010103.

[86] C. Castro-Cabrera and I. Medina-Bulo, "An approach to metamorphic testing for WS-BPEL compositions," in *Proc. Int. Conf. e-Bus.*, Jul. 2011, pp. 1–6.

[87] C. Castro-Cabrera and I. Medina-Bulo, "Application of metamorphic testing to a case study in web services compositions," in *E-Business and Telecommunications* (ser. Communications in Computer and Information Science), M. Obaidat, J. Sevillano, and J. Filipe, Eds., Berlin, Germany: Springer, 2012, vol. 314, pp. 168–181. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-35755-8_13.

[88] OASIS: Web Services Business Process Execution Language. (2007, Apr.). [Online]. Available: 2.0. http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html.

[89] Z. Q. Zhou, T. H. Tse, F.-C. Kuo, and T. Y. Chen, "Automated functional testing of web search engines in the absence of an oracle," Dept. Comput. Sci., The Univ. Hong Kong, Tech. Rep. TR-2007-06, 2007.

[90] Z. Q. Zhou, S. Xiang, and T. Y. Chen, "Metamorphic testing for software quality assessment: A study of search engines," *IEEE Trans. Softw. Eng.*, 2015, Doi: 10.1109/TSE.2015.2478001.

[91] J. Mayer and R. Guderlei, "On random testing of image processing applications," in *Proc. 6th Int. Conf. Quality Softw.*, Oct. 2006, pp. 85–92.

[92] R. Guderlei and J. Mayer. (2007). Towards automatic testing of imaging software by means of random and metamorphic testing. *Int. J. Softw. Eng. Knowl. Eng.* [Online]. *17(06)*, pp. 757–781. Available: http://www.worldscientific.com/doi/abs/10.1142/S0218194007003471.

[93] W. K. Chan, J. C. F. Ho, and T. H. Tse, "Piping classification to metamorphic testing: An empirical study towards better effectiveness for the identification of failures in mesh simplification programs," in *Proc. 31st Annu. Int. Comput. Softw. Appl. Conf.*, vol. 1, pp. 397–404, Jul. 2007.

[94] W. K. Chan, J. C. F. Ho, and T. H. Tse. (2010, Jun.). Finding failures from passed test cases: Improving the pattern classification approach to the testing of mesh simplification programs. *Softw. Testing, Verification Rel. J.* [Online]. *20(2)*, pp. 89–120. Available: http://dx.doi.org/10.1002/stvr.v20:2.

[95] R. Just and F. Schweiggert, "Evaluating testing strategies for imaging software by means of mutation analysis," in *Proc. Int. Conf. Softw. Testing, Verification Validation Workshops*, Apr. 2009, pp. 205–209.

[96] T. Jameel, L. Mengxiang, and C. Liu, "Test oracles based on metamorphic relations for image processing applications," in *Proc. 16th IEEE/ACIS Int. Conf. Softw. Eng., Artif. Intell., Netw. Parallel/Distrib. Comput.*, Jun. 2015, pp. 1–6.

[97] T. H. Tse, S. S. Yau, W. K. Chan, H. Lu, and T. Y. Chen, "Testing context-sensitive middleware-based software applications," in *Proc. 28th Annu. Int. Comput. Softw. Appl. Conf.*, vol. 1, pp. 458–466, Sep. 2004.

[98] W. K. Chan, T. Y. Chen, H. Lu, T. H. Tse, and S. S. Yau, "A metamorphic approach to integration testing of context-sensitive middleware-based applications," in *Proc. 5th Int. Conf. Quality Softw.*, Sep. 2005, pp. 241–249.

[99] W. K. Chan, T. Y. Chen, S. C. Cheung, T. H. Tse, and Z. Zhang, " Towards the testing of power-aware software applications for wireless sensor networks," in *Ada Europe 2007 - Reliable Software Technologies*, N. Abdennadher and F. Kordon, Eds., Berlin, Germany: Springer, 2007, vol. 4498, pp. 84–99. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-73230-3_7.

[100] F.-C. Kuo, T. Y. Chen, and W. K. Tam, "Testing embedded software by metamorphic testing: A wireless metering system case study," in *Proc. IEEE 36th Conf. Local Comput. Netw.*, Oct. 2011, pp. 291–294.

[101] M. Jiang, T. Y. Chen, F.-C. Kuo, and Z. Ding, "Testing central processing unit scheduling algorithms using metamorphic testing," in *Proc. 4th IEEE Int. Conf. Softw. Eng. Service Sci.*, May 2013, pp. 530–536.

[102] K. Y. Sim, W. K. S. Pao, and C. Lin, "Metamorphic testing using geometric interrogation technique and its application," in *Proc. 2nd Int. Conf. Elect. Eng./Electron., Comput., Telecommun., Inform. Technol.*, 2005, pp. 91–95.

[103] T. Y. Chen, F.-C. Kuo, H. Liu, and S. Wang, " Conformance testing of network simulators based on metamorphic testing technique," in *Formal Techniques for Distributed Systems*, D. Lee, A. Lopes, and A. Poetzsch-Heffter, Eds., Berlin, Germany: Springer, vol. 5522, pp. 243–248, 2009. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-02138-1_19.

[104] OMNeT++ system. (2015, Dec.). [Online]. Available: http://www.omnetpp.org.

[105] J. Ding, T. Wu, D. Xu, J. Q. Lu, and X. Hu, "Metamorphic testing of a Monte Carlo modeling program," in *Proc. 6th Int. Workshop Autom. Softw. Test*, 2011, pp. 1–7. [Online]. Available: http://doi.acm.org/10.1145/1982595.1982597.

[106] A. Nuñez and R. M. Hierons. (2014). A methodology for validating cloud models using metamorphic testing. *Ann. Telecommun. – annales des télécommunications* [Online]. pp. 1–9. Available: http://dx.doi.org/10.1007/s12243-014-0442-7.

[107] A. Nuñez, J. L. Vazquez-Poletti, A. C. Caminero, G. G. Castañe, J. Carretero, and I. M. Llorente. (2012). icancloud: A flexible and scalable cloud infrastructure simulator. *J. Grid Comput.*, 10(1), pp. 185–209. [Online]. Available: http://dx.doi.org/10.1007/s10723-012-9208-5.

[108] P. C. Cañizares, A. Núñez, M. Núñez, and J. J. Pardo, "A methodology for designing energy-aware systems for computational science," *Procedia Comput. Sci.*, vol. 51, pp. 2804–2808, 2015, (Int. Conf. Comput. Sci.) [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1877050915012466.

[109] X. Xie, J. Ho, C. Murphy, G. Kaiser, B. Xu, and T. Y. Chen, "Application of metamorphic testing to supervised classifiers," in *Proc. 9th Int. Conf. Quality Softw.*, Aug. 2009, pp. 135–144.

[110] J. E. Gewehr, M. Szugat, and R. Zimmer. (2007, Feb.). Bioweka—extending the weka framework for bioinformatics. *Bioinformatics*. [Online]. *23(5)*, pp. 651–653. Available: http://dx.doi.org/10.1093/bioinformatics/btl671.

[111] Z. Jing, H. Xuegang, and Z. Bin. (2011). An evaluation approach for the program of association rules algorithm based on metamorphic relations. *J. Electron. (China)* [Online]. *28(4-6)*, pp. 623–631. Available: http://dx.doi.org/10.1007/s11767-012-0743-9.

[112] L. L. Pullum and O. Ozmen, "Early results from metamorphic testing of epidemiological models," in *Proc. ASE/IEEE Int. Conf. BioMedical Comput.*, Dec. 2012, pp. 62–67.

[113] A. Ramanathan, C. A. Steed, and L. L. Pullum, "Verification of compartmental epidemiological models using metamorphic testing, model checking and visual analytics," in *Proc. ASE/IEEE Int. Conf. BioMed. Comput.*, Dec. 2012, pp. 68–73.

[114] S. Beydeda, "Self-metamorphic-testing components," in *Proc. 30th Annu. Int. Comput. Softw. Appl. Conf.*, vol. 2, pp. 265–272, Sep. 2006.

[115] X. Lu, Y. Dong, and C. Luo, "Testing of component-based software: A metamorphic testing methodology," in *Proc. Int. Conf. Ubiquitous Intell. Comput. Int. Conf. Auton. Trusted Comput.*, Oct. 2010, pp. 272–276.

[116] T. Y. Chen, J. Feng, and T. H. Tse, "Metamorphic testing of programs on partial differential equations: A case study," in *Proc. 26th Int. Comput. Softw. Appl. Conf. Prolonging Softw. Life: Develop. Redevelop.*, 2002, pp. 327–333. [Online]. Available: http://dl.acm.org/citation.cfm?id=645984.675903.

[117] C. Aruna and R. S. R. Prasad, "Metamorphic relations to improve the test accuracy of multi precision arithmetic software applications," in *Proc. Int. Conf. Adv. Comput., Commun. Informat.*, Sep. 2014, pp. 2244–2248.

[118] Q. Tao, W. Wu, C. Zhao, and W. Shen, "An automatic testing approach for compiler based on metamorphic testing technique," in *Proc. 17th Asia Pacific Softw. Eng. Conf.*, Nov. 2010, pp. 270–279.

[119] V. Le, M. Afshari, and Z. Su, "Compiler validation via equivalence modulo inputs," in *Proc. 35th ACM SIGPLAN Conf. Program. Language Des. Implementation*, 2014, pp. 216–226. [Online]. Available: http://doi.acm.org/10.1145/2594291.2594334.

[120] T. Y. Chen, F.-C. Kuo, and Z. Q. Zhou, "An effective testing method for end-user programmers," in *Proc. 1st Workshop End-User Softw. Eng.*, 2005, pp. 1–5. [Online]. Available: http://doi.acm.org/10.1145/1082983.1083236.

[121] K. Y. Sim, C. S. Low, and F.-C. Kuo, "Detecting faults in technical indicator computations for financial market analysis," in *Proc. 2nd Int. Conf. Inform. Sci. Eng.*, Dec. 2010, pp. 2749–2754.

[122] (2016). MetaTrader 4 Trading Terminal. [Online]. Available: http://www.metaquotes.net/en/metatrader4/trading_terminal.

[123] S. Yoo, "Metamorphic testing of stochastic optimisation," in *Proc. 3rd Int. Conf. Softw. Testing, Verification, Validation Workshops*, Apr. 2010, pp. 192–201.
[124] Y. Yao, S. Huang, and M. Ji, "Research on metamorphic testing for oracle problem of integer bugs," in *Proc. 4th Int. Conf. Adv. Comput. Sci. Inf. Eng.*, ser. Advances in Intelligent and Soft Computing, 2012, vol. 168, pp. 95–100. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-30126-1_16.
[125] Y. Yao, C. Zheng, S. Huang, and Z. Ren, "Research on metamorphic testing: A case study in integer bugs detection," in *Proc. 4th Int. Conf. Intell. Syst. Design Eng. Appl., 2013*, Nov. 2013, pp. 488–493.
[126] Z. Hui, S. Huang, Z. Ren, and Y. Yao, "Metamorphic testing integer overflow faults of mission critical program: A case study," *Math. Problems Eng.*, vol. 2013, pp. 1–6, 2013.
[127] G. Batra and G. Singh, "An automated metamorphic testing technique for designing effective metamorphic relations," *Contemporary Computing* (ser. Communications in Computer and Information Science), M. Parashar, D. Kaushik, O. Rana, R. Samtaney, Y. Yang, and A. Zomaya, Eds., Berlin, Germany: Springer, 2012, vol. 306, pp. 152–163. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-32129-0_20.
[128] C. Sun, Z. Wang, and G. Wang. (2014). A property-based testing framework for encryption programs. *Frontiers Comput. Sci.* [Online]. *8(3)*, pp. 478–489. Available: http://dx.doi.org/10.1007/s11704-014-3040-y.
[129] C. Aruna and R. S. R. Prasad, "Adopting metamorphic relations to verify non-testable graph theory algorithms," in *Proc. 2nd Int. Conf. Adv. Comput. Commun. Eng.*, May 2015, pp. 673–678.
[130] M. Lindvall, D. Ganesan, R. Ardal, and R. Wiegand, "Metamorphic model-based testing applied on NASA DAT—An experience report," in *Proc. 37th IEEE Int. Conf. Softw. Eng.*, vol. 2, pp. 129–138, May 2015.
[131] T. Y. Chen, T. H. Tse, and Z. Q. Zhou, "Fault-based testing in the absence of an oracle," in *Proc. 25th Annu. Int. Comput. Softw. Appl. Conf.*, 2001, pp. 172–178.
[132] C. Cadar and K. Sen. (Feb 2013). Symbolic execution for software testing: Three decades later. *Commun. ACM* [Online]. *56(2)*, pp. 82–90. Available: http://doi.acm.org/10.1145/2408776.2408795.
[133] I. Erete and A. Orso, "Optimizing constraint solving to better support symbolic execution," in *Proc. Workshop Constraints Softw. Testing, Verification, Anal.*, 2011, pp. 310–315.
[134] T. Y. Chen, T. H. Tse, and Z. Q. Zhou, "Semi-proving: An integrated method based on global symbolic evaluation and metamorphic testing," in *Proc. ACM SIGSOFT Int. Symp. Softw. Testing Anal.*, 2002, pp. 191–195. [Online]. Available: http://doi.acm.org/10.1145/566172.566202.
[135] G. Dong, S. Wu, G. Wang, T. Guo, and Y. Huang, "Security assurance with metamorphic testing and genetic algorithm," in *Proc. IEEE/WIC/ACM Int. Conf. Web Intell. Intell. Agent Technol.*, vol. 3, pp. 397–401, Aug. 2010.
[136] X. Xie, W. E. Wong, T. Y. Chen, and B. Xu, "Spectrum-based fault localization: Testing oracles are no longer mandatory," in *Proc. 11th Int. Conf. Quality Softw.*, Jul. 2011, pp. 1–10.
[137] X. Xie, W. E. Wong, T. Y. Chen, and B. Xu. (2013). Metamorphic slice: An application in spectrum-based fault localization," *Inform. Softw. Technol.* [Online]. *55(5)*, pp. 866–879. Available: http://www.sciencedirect.com/science/article/pii/S0950584912001759.
[138] (2016). Siemens Suite. [Online]. Available: http://sir.unl.edu/portal/bios/tcas.php#siemens.
[139] Y. Lei, X. Mao, and T. Y. Chen, "Backward-slice-based statistical fault localization without test oracles," in *Proc. 13th Int. Conf. Quality Softw.*, Jul. 2013, pp. 212–221.
[140] Y. Lei, X. Mao, Z. Dai, and C. Wang, "Effective statistical fault localization using program slices," in *Proc. Comput. Softw. Appl. Conf.*, Jul. 2012, pp. 1–10.
[141] P. Rao, Z. Zheng, T. Y. Chen, N. Wang, and K. Cai, "Impacts of test suite's class imbalance on spectrum-based fault localization techniques," in *Proc. 13th Int. Conf. Quality Softw.*, Jul. 2013, pp. 260–267.
[142] C. Aruna and R. S. R. Prasad, "Testing approach for dynamic web applications based on automated test strategies," in *ICT and Critical Infrastructure: Proc. 48th Annu. Convention of Comput. Soc. India—Vol II*, ser. Advances in Intelligent Systems and Computing, vol. 249, pp. 399–410, 2014. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-03095-1_43.
[143] S. Artzi, A. Kiezun, J. Dolby, F. Tip, D. Dig, A. Paradkar, and M. D. Ernst, "Finding bugs in dynamic web applications," in *Proc. Int. Symp. Softw. Testing Anal.*, 2008, pp. 261–272. [Online]. Available: http://doi.acm.org/10.1145/1390630.1390662.
[144] H. Liu, I. I. Yusuf, H. W. Schmidt, and T. Y. Chen, "Metamorphic fault tolerance: An automated and systematic methodology for fault tolerance in the absence of test oracle," in *Proc. 36th Int. Conf. Softw. Eng.*, 2014, pp. 420–423. [Online]. Available: http://doi.acm.org/10.1145/2591062.2591109.
[145] H. Jin, Y. Jiang, N. Liu, C. Xu, X. Ma, and J. Lu, "Concolic metamorphic debugging," in *Proc. IEEE 39th Annu. Comput. Softw. Appl. Conf.*, vol. 2, pp. 232–241, Jul. 2015.
[146] H. Liu, F.-C. Kuo, and T. Y. Chen, "Teaching an end-user testing methodology," in *Proc. 23rd IEEE Conf. Softw. Eng. Educ. Training*, Mar. 2010, pp. 81–88.
[147] S. Yoo and M. Harman. (2012). Regression testing minimization, selection and prioritization: A survey. *Softw. Testing, Verification Rel.* [Online]. *22(2)*, pp. 67–120. Available: http://dx.doi.org/10.1002/stvr.430.
[148] M. D. Ernst, J. Cockrell, W. G. Griswold, and D. Notkin, "Dynamically discovering likely program invariants to support program evolution," in *Proc. 21st Int. Conf. Softw. Eng.*, 1999, pp. 213–224. [Online]. Available: http://doi.acm.org/10.1145/302405.302467.

Sergio Segura received the PhD degree in software engineering (with honours) from Seville University, Sevilla, Spain, where he currently works as a senior lecturer. His research interests include software testing, software variability and search-based software engineering. He has co-authored some highly cited papers as well as tools used by universities and companies in various countries. He also serves regularly as a reviewer for international journals and conferences.

Gordon Fraser received the PhD degree from the Graz University of Technology, Graz, Austria, in 2007, and worked as a postdoctoral researcher at Saarland University, Germany. He is a senior lecturer in computer science at the University of Sheffield, United Kingdom. His research is on improving software quality and programmer productivity. He is the chair of the steering committee of the International Conference on Software Testing, Verification, and Validation, and is regular organising- and programme-committee member of software engineering conferences and workshops. His work on software testing has achieved wide recognition both in research (e.g., DFG and EPSRC grants, ACM SIGSOFT distinguished and best paper awards at FSE, ISSTA, ASE, SSBSE, and GECCO) as well as industry (e.g., Google Focused Research Award or Microsoft Software Engineering Innovation Foundation Award).

Ana B. Sánchez is currently working toward the PhD degree at the Applied Software Engineering Research Group, University of Seville where she received the MSc degree. Her research focuses on automating the detection of faults in highly configurable systems.

Antonio Ruiz-Cortés received the PhD and MSc degrees in computer science from the University of Seville, Spain. He is an accredited full professor and the head of the Applied Software Engineering Research Group, University of Seville. His research interests include the areas of service oriented computing, software variability, software testing, and business process management.