Software Qual J
https://doi.org/10.1007/s11219-017-9392-4

A mapping study on testing non-testable systems

Krishna Patel[1] · Robert M. Hierons[1]

Abstract The terms "Oracle Problem" and "Non-testable system" interchangeably refer to programs in which the application of test oracles is infeasible. Test oracles are an integral part of conventional testing techniques; thus, such techniques are inoperable in these programs. The prevalence of the oracle problem has inspired the research community to develop several automated testing techniques that can detect functional software faults in such programs. These techniques include N-Version testing, Metamorphic Testing, Assertions, Machine Learning Oracles, and Statistical Hypothesis Testing. This paper presents a Mapping Study that covers these techniques. The Mapping Study presents a series of discussions about each technique, from different perspectives, e.g. effectiveness, efficiency, and usability. It also presents a comparative analysis of these techniques in terms of these perspectives. Finally, potential research opportunities within the non-testable systems problem domain are highlighted within the Mapping Study. We believe that the aforementioned discussions and comparative analysis will be invaluable for new researchers that are attempting to familiarise themselves with the field, and be a useful resource for practitioners that are in the process of selecting an appropriate technique for their context, or deciding how to apply their selected technique. We also believe that our own insights, which are embedded throughout these discussions and the comparative analysis, will be useful for researchers that are already accustomed to the field. It is our hope that the potential research opportunities that have been highlighted by the Mapping Study will steer the direction of future research endeavours.

Keywords Software testing · Oracle problem · Non-testable · Test oracles · Mapping study · Survey

✉ Krishna Patel
cspgkkp@brunel.ac.uk

Robert M. Hierons
rob.hierons@brunel.ac.uk

[1] Department of Computer Science, Brunel University, Uxbridge, Middlesex, UB8 3PH, UK

Published online: 24 November 2017

1 Introduction

In software testing, a test input is generated for the System Under Test (SUT), an expected test outcome is determined for this test input, and the SUT is executed with this test input to obtain an output. This output is finally compared to the expected test outcome. If this comparison reveals any discrepancies, then the SUT is deemed to be faulty. The mechanism that is responsible for predicting the expected test outcome and performing this comparison task is called an oracle. Software testing is based on the assumption that an oracle is always available (Davis and Weyuker 1981). The terms "Non-testable systems" (Davis and Weyuker 1981) and "oracle problem" (Liu et al. 2014a) are interchangeably used to describe situations in which it is infeasible to apply oracles. Conventional testing techniques are ineffective under such circumstances.

Several automated testing techniques that can detect functional software faults in non-testable systems have been proposed. A sizeable amount of research has been conducted on these techniques in the context of the oracle problem. Our Mapping Study endeavours to collect, collate, and synthesise this research to satisfy three major objectives. The first objective is to present a series of discussions about each of these techniques, from different perspectives, e.g. effectiveness, usability, and efficiency. The second objective is to perform a series of comparisons between these techniques, based on the above perspectives. The final objective is to identify research opportunities.

This paper begins by outlining the Review Protocol (see Section 2). A series of discussions revolving around each testing technique are presented across Sections 3 to 7. These techniques are compared in Section 8. Potential future research directions are also presented across these sections. Additionally, Section 9 discusses related work. Threats to validity are discussed in Section 10, and our conclusions are outlined in Section 11.

2 Review protocol

To conduct our Mapping Study, we first defined a Review Protocol, based on the guidelines of Kitchenham (2007), Popay et al. (2006), Higgins and Green (2011), and Shepperd (2013). This section presents our Review Protocol. In particular, Sections 2.1, 2.2, 2.3 and 2.4 outline the scope, search procedure, data extraction approach, and quality appraisal methodology, respectively. Finally, a brief overview of the synthesis, which forms the majority of the remainder of this paper, is presented in Section 2.5.

2.1 Scope

The scope of this Mapping Study was originally *automated testing and debugging* techniques that have been designed to detect *functional software* faults in *non-testable systems*. Our Review Protocol (presented in Sections 2.2 to 2.4) is based on this scope. We decided to narrow the scope of our synthesis (i.e. Sections 2.5 to 8) to enhance the focus of the Mapping Study. In particular, our revised scope does not consider debugging techniques. We realised that Specification-based Testing and Model-based Testing depend on the availability of a specification or model, and that the oracle problem implies that these are not available. To that end, we also decided to omit these techniques from the scope of our synthesis.

2.2 Search

A paper is considered to be relevant if it adheres to the Inclusion and Exclusion Criteria listed in Table 1. To check a paper against these criteria, we adopted an iterative process,

Table 1 Relevance inclusion and exclusion criteria

Issue	Criteria
Problem Domain	*The targeted problem domain/context in which testing is undertaken must be non-testable.*
	The problem itself must revolve around the lack of a mechanism to judge the correctness of an output.
	The existence of the non-testable aspect must not be considered to be a fault in itself.
	The non-testable characteristic experienced in the SUT must arise from the software.
	The types of faults considered by the paper must be software faults.
	The quality attribute of the system being tested must be functional correctness.
Solution Space	*The paper must include some sort of solution to the problem, e.g. a testing technique.*
	The primary solution to the problem must revolve around an automated fault detection mechanism.
	The solutions must fall under the domains of either testing or debugging.
Paper Type	*Journal articles, conference proceedings, technical reports, book chapters, and magazines must be included.*
	Papers that have a broad focus, e.g. frameworks or systematic reviews must contribute a relatively substantial amount of relevant content. For example, a paper is not deemed to be relevant if all of its relevant material is comprised of a short aside.
	Duplicates must be excluded. We consider rewrites and preliminary/older versions of the same papers to be duplicates. We also consider journal papers that extend conference papers to be duplicates, as well as published chapters of theses. Preference is given to published over non-published papers, the most up-to-date version, and the paper from the most reputable source. If both papers are published in reputable journals, the most detailed one is taken, and in the case that they are precise duplicates of each other, an arbitrary choice is made.
	The paper must be written in English.
	The paper must be accessible.
	The paper must have been published before mid 2014.

where successive iterations checked the paper in escalating levels of detail (Shepperd 2013) against the Inclusion and Exclusion Criteria; if enough evidence could be accrued during an early iteration to classify the paper, then the process terminated prematurely. The iterations were as follows: {title}, {abstract, introduction, conclusion}, {the entire paper}. We conducted a search in mid 2014 to find relevant papers (that were available before and during mid 2014). We achieved this by applying several search methods in parallel and iteratively, and checking the relevance of each search result returned by these search methods, by using the aforementioned iterative process. The remainder of this section outlines these search methods.

One of our methods included using the search strings listed in Table 2 to query six research repositories: Brunel University Summon, ScienceDirect (using the Computer Science Discipline filter), ACM (queried using Google's "site:" function), IEEE, Google (twice — with the filter on, and off), and Citeseerx (each search term prefixed with "text:"). Let $Results_{SS}^{RR}$ denote the papers that were returned by a research repository, RR, in response to a search string, SS. It would have been infeasible to manually check the relevance of all of the papers in $Results_{SS}^{RR}$. Thus, we used the following terminating condition: the first occurrence of 50 consecutive irrelevant results. We retained papers that were found to be relevant before the terminating condition was satisfied.

During the search process, we became aware of several techniques that had been used to solve the oracle problem. The authors postulated that other studies on these techniques in the context of the oracle problem may also have been conducted. Thus, a specialised search string for each technique was prepared; these search strings supplemented those in Table 2.

Every paper in the reference list of each relevant paper was also checked for relevance. Again, we retained papers that were found to be relevant.

We compiled a list of all of the authors that had contributed to the papers that were deemed to be relevant. Each of these authors had at least one list of their publications

Table 2 Search strings

Search Strings
((Stochastic OR (Non-deterministic OR nondeterministic OR "non deterministic" OR non-determinism OR nondeterminism OR "non determinism")) AND (System OR Software OR Program OR Application OR Algorithm) AND Testing
((Stochastic OR (Non-deterministic OR nondeterministic OR "non deterministic" OR non-determinism OR nondeterminism OR "non determinism")) AND (System OR Software OR Program OR Application OR Algorithm) AND (("Check" OR "Checking") OR ("Verification" OR "Verify"))
((Stochastic OR (Non-deterministic OR nondeterministic OR "non deterministic" OR non-determinism OR nondeterminism OR "non determinism")) AND (System OR Software OR Program OR Application OR Algorithm) AND ("Fault Localisation" OR "Fault Localization")
("Random output" OR "Randomised output" OR "Randomized output" OR "Randomized algorithm" OR "Randomised algorithm") AND (Systems OR Software OR Programs OR Applications OR Algorithms) AND Testing
("Probabilistic System" OR "Probabilistic Program" OR "Probabilistic algorithm") AND (("Check" OR "Checking") OR ("Verification" OR "Verify"))
(("NonTestable" OR "Non-Testable" OR "Non Testable") OR ("Oracle Problem" OR "no oracle") OR ("Pseudo-oracle" OR "Pseudo oracle")) AND (Testing OR (("Check" OR "Checking") OR ("Verification" OR "Verify")) OR ("fault localisation" OR "Fault localization") OR ("Debugging" OR "Debug") OR ("Fault detection" OR "Failure detection" OR "Mutant detection" OR "Defect detection" OR "Detecting Faults" OR "Detecting Failures" OR "Detecting Mutants" OR "Detecting Defects") OR ("Validating" OR "Validate"))

publicly available. Examples of such lists include: author's personal web page, CV, DBLP, Google Scholar, and the repository the author's study was originally discovered in. We selected one list per author, based on availability and completeness, and checked all of the publications on this list for relevance.

Finally, all of the authors were emailed a list of their papers that had been discovered by the search, accompanied with a request for confirmation regarding the completeness of the list. This enabled the procurement of cutting edge works in progress, and also reduced publication bias (Kitchenham 2007).

The various search methods described above led to the discovery of several papers that we did not have access to. We were able to obtain some of these papers by contacting the authors. The rest of these papers were omitted from the Mapping Study. The search methods also returned what we believed were duplicate research papers. The authors of these papers were asked to confirm our suspicions, and duplicates were removed.

In total, our search methods collectively procured 141 papers.

2.3 Data extraction

We used the data extraction form in Table 3 to capture data, from relevant papers, that was necessary to appraise study quality and address the research aims. Unfortunately, many papers did not contain all of the required data; thus requests were sent to authors to obtain missing data. Where this approach was unsuccessful, assumptions were made based on the paper and the author's other work. For example, if they had not reported the number of mutants used, but tended to use 1000+ in other papers, one can assume a significant number of mutants were used in the study.

Table 3 Data extraction form

Data Extraction Form	
ID:	Paper Title:
Question	Answer
What evidence is there of there being a sufficient amount of relevant information in the paper to make a meaningful contribution to the Mapping Study's findings?	
What evidence is there to suggest the parameters of the experiment were representative and were they adequate described?	
What evidence is there to suggest the experimental set-up, conduct and experiment output analysis was appropriate, robust and unbiased?	
Have adverse effects been reported, and if so, how were they mitigated?	
Are the arguments compelling, critical and supported by internal and external evidence?	
Executive Summary:	
Noteworthy points made about Technique 1:	
Noteworthy points made about Technique 2:	
Noteworthy points made about Technique n:	

2.4 Quality

Our quality instrument is presented in Table 4. Each relevant paper was checked against this quality instrument. Papers that were found to have severe methodical flaws, and to have taken minimal steps to mitigate bias were deemed to be of low quality. Relatively

Table 4 Quality instrument

Questions	Answers
Q1. Does the SUT have the non-testable characteristics that are being studied? *This question is not applicable to "Demonstration Papers" or "Extreme Depth of Analysis Papers" See below for their definitions.*	
Q2. Can you identify sources of potential bias in the paper, and are there any elements of the study design that can minimise bias or justify leaving the source of bias unchecked? *This question is not applicable to "Demonstration Papers" or "Extreme Depth of Analysis Papers" See below for their definitions.*	
Q3. If the author used measures or made inferences that are sensitive to the number of mutants, did they use an appropriate number of mutants? *This question is not applicable to "Demonstration Papers" or "Extreme Depth of Analysis Papers" See below for their definitions.*	
Q4. If the author used measures or made inferences that are sensitive to the number of test cases, did they use an appropriate number of test cases? *This question is not applicable to "Demonstration Papers" or "Extreme Depth of Analysis Papers" See below for their definitions.*	
Q5. If the author used measures or made inferences that are sensitive to the number of participants, did they use an appropriate number of participants? *This question is not applicable to "Demonstration Papers" or "Extreme Depth of Analysis Papers" See below for their definitions.*	
Q6. Were the authors' arguments and inferences backed up by internal and external evidence?	
Q7. Did the authors' use of language suggest they were biased towards a specific technique/findings, e.g. were positive comments given about the comparison intervention or negative comments about the authors own technique?	
Q8. Was the authors' study novel? e.g. Extensions, using participants, including non-effectiveness measures etc.	
Comments:	

Definitions

A demonstration paper is one that conducts an experiment, but not for the purpose of assessing the quality of the technique, but rather, to illustrate how it works/that it does work. One of the key features of a demonstration paper is that the experiment is not set-up to be rigorous, but to instead be an effective communication tool - i.e. simple and intuitive.

"Extreme depth of analysis" papers are papers where either the results themselves have fairly complex associations amongst each other, e.g. every pairwise combination of each test case and mutant has been individually presented, or if the author has a fairly extensive discussion about all of the individual cases.

little research has been conducted on the oracle problem; thus, many relevant studies are exploratory. Certain study design choices may have been unavoidable in such studies, and may cause a quality instrument to label these studies as low quality. This means that these valuable studies could be rejected, despite the fact that they may have been at the highest attainable quality at the time. To account for this, papers that were deemed to be of low quality, were only discarded if they did not make a novel contribution. This led to the elimination of 4 papers. Thus a total of 137 papers were deemed to be suitable for our synthesis.

2.5 Synthesis overview

Synthesis involves analysing and explaining the data that was obtained by the data extraction form to address the research aims. Narrative Synthesis was used because it is ideal for theory building (Shepperd 2013) and explanations. The synthesis was conducted according to the guidelines of Popay et al. (2006), Cruzes and Dyba (2011), Da Silva et al. (2013), and Barnett-Page and Thomas (2009). See Sections 3 to 8.

The Mapping Study process revealed that five umbrella testing techniques have been developed to alleviate the oracle problem — N-version Testing, Metamorphic Testing, Assertions, Machine Learning, and Statistical Hypothesis Testing. Thus, our synthesis focuses on these techniques. The research community has conducted a different amount of research on each technique, in the context of the oracle problem. For example, Metamorphic Testing has received more attention than any other testing technique. Naturally, the amount of attention that is afforded to each technique, by our synthesis, was determined by the amount of research that was conducted on that technique.

The disproportionate attention that has been given to Metamorphic Testing suggests that this technique may have numerous interesting research avenues. Although less attention has been afforded to the other techniques, they are known to be effective for some situations in which Metamorphic Testing is not (see Section 8). Thus, the number of pages does not reflect how promising they are. However, it does mean that it is unlikely that all of the useful research avenues that are associated with these techniques have been explored.

Sections 3–7 present a series of discussions about each technique, and Section 8 compares these techniques. The discussions pertaining to each technique are organised into a set of high level issues, e.g. effectiveness, efficiency, and usability. Some terms that are used to describe certain issues by one research community may be used to describe different issues by other research communities. We would therefore like to clarify how such terms are used in this paper; in particular efficiency and cost. Efficiency is used to describe the amount of computational resources that are consumed or the amount of time required to perform a task, whilst cost is used in reference to monetary costs. Although effort/manual labour can be discussed in the context of cost, effort/manual labour is presented as a usability issue in this paper.

3 N-version testing

Let S be the SUT. Another system, S_R, is said to be a reference implementation (RI) of S, if it implements some of the same functionality as S. In N-version Testing, S and S_R are executed with the same test case, such that this test case executes the common functionality in these systems. The outputs of S and S_R that result from these executions are compared. N-version Testing reports a failure, if these outputs differ (Weyuker 1982). If one has access to multiple RIs, then this process can be repeated once for each RI.

N-version Testing was originally developed to alleviate the oracle problem. One form of oracle problem includes situations where the test outcome is unpredictable. Such an oracle problem can arise if the SUT has been designed to discover new knowledge (Weyuker 1982), e.g. machine learning algorithms. Since an RI mimics the SUT to generate its own output, N-Version Testing does not require the tester to have prior knowledge about the test outcome. This makes it viable for such oracle problems.

3.1 Effectiveness of N-version testing

N-version Testing is fundamentally a black-box testing technique. It's therefore not surprising that some have found that N-version Testing cannot test the flow of events (Nardi and Delamaro 2011), and cannot detect certain fault types, e.g. coincidental correctness[1] (Brilliant et al. 1990), since white-box oracle information is necessary to achieve these feats. Let S be a system. In the future, S may be modified due to software maintenance. S' denotes the modified version of S. Faults may be introduced into S' during maintenance. Spectrum-based Fault Localisation is a debugging technique that represents the system's execution trace as program spectra. Tiwari et al. (2011) suggested using Spectrum-based Fault Localisation to obtain the program spectras of S and S', and comparing these program spectras. Disparities between these program spectra can be an indication of a fault in S'. In their approach, S' is essentially the SUT, and S acts as a reference implementation. Thus, their approach can be perceived to be a modified version of N-version Testing, in which program spectra are compared instead of outputs. Since program spectra can represent event flows, this modified version of N-version Testing may be able to test the flow of events. However, there is little evidence to suggest that this approach can generalise outside of regression testing. Thus, we believe that feasibility studies that explore the use of this approach in other contexts would be valuable.

Let S denote the SUT and S_R be a reference implementation of S, such that S and S_R have faults that result in the same failed outputs, S^o and S_R^o , respectively. Since N-version Testing detects a fault by checking $S^o \neq S_R^o$ (Manolache and Kourie 2001), it cannot detect these faults. This is referred to as a correlated failure. Numerous guidelines for reducing the likelihood of correlated failures are available. The remainder of this section explores these guidelines.

A fault is more likely to be replicated in both S and S_R if the same team develop both, because they might be prone to making the same types of mistakes (Murphy et al. 2009a). Thus, one guideline includes using independent teams for each system (Manolache and Kourie 2001), e.g. using 3rd party software as S_R. However, this does not eliminate the problem completely because independent development teams can also make the same mistakes (Murphy et al. 2009a). This could be because certain systems are susceptible to particular fault types. Thus, another guideline involves diversifying the systems to reduce the overlap of possible fault types across systems (Manolache and Kourie 2001). This can be achieved by using different platforms, algorithms (Lozano 2010), design methods, software architectures, and programming languages (Manolache and Kourie 2001) for each system. For example, pointer related errors are less likely to lead to correlated failures if S and S_R are encoded in C++ and Java, respectively.

[1]Coincidental correctness describes a situation in which a faulty program state manifests during an execution of the system, but despite this, the system produces an output that could have been produced by a correct version of the system.

The third guideline we consider revolves around manipulating the test suite. Some test inputs lead to correlated failures, and others do not (Brilliant et al. 1990). Thus, the chance of detecting a fault depends on the ratio of inputs that lead to a correlated failure (CF) to inputs that do not (we refer to non-correlated failures as standard failures (SF)). Since multiple faults may collectively contribute to populating CF and diminishing SF (Brilliant et al. 1990), one could adopt a strategy of re-executing the test suite when a fault is removed because this may improve the $CF : SF$ ratio. To demonstrate, let f_1 and f_2 represent two faults in the SUT, and $\{1, 2, 3, 4, 5\}$ be a set of inputs that lead to a correlated failure as a result of f_1. Further suppose that $\{5, 6\}$ is the set of inputs that can detect f_2. Since f_1 causes 5 to lead to a correlated failure, only input 6 can detect f_2; thus by removing f_1, the number of inputs that can be used to detect f_2 doubles, since 5 would no longer lead to a correlated failure.

Although the guidelines discussed above (i.e. using independent development teams, diversifying the systems, and test suite manipulation) can reduce the number of correlated failures, the extent to which they do varies across systems. This is because different systems have outputs with different cardinalities,[2] which have been observed to influence the incidence of correlated failures (Brilliant et al. 1990).

3.2 Usability of N-version testing

The only manual tasks in N-version Testing are procuring RIs and debugging; thus discussions regarding usability will revolve around these issues. This section discusses the former, and the latter is covered in Section 3.3.

At its inception, the recommended method of procuring RIs for the purpose of N-version Testing was development (Weyuker 1982). Developing RIs can require substantial time and effort (Hummel et al. 2006). Many solutions have been proposed, that might reduce the labour intensiveness of this task. For example, Davis and Weyuker (1981) recognised that the performance of an RI is not important, because it is not intended to be production quality code. They also realised that program that are written in High Level Programming Languages have poorer runtime efficiency, but are quicker and easier to develop (Davis and Weyuker 1981). To that end, they suggest using such languages for the development of RIs. Similarly, we suspect that it might be possible to sacrifice other quality attributes, to make RI development faster and easier.

Solutions that can eliminate development effort completely have also been proposed. For example, it has been reported that the previous versions of the same system (Zhang et al. 2009) (this approach has been widely adopted in practice), or other systems that implement the same functionality (Chan et al. 2009) could be used as RIs. RIs could also be automatically generated, e.g. through Testability Transformations (McMinn 2009). Testability Transformations automatically generate RIs by modifying the original system's source code, O, into a syntactically different version, M, such that M and O are observationally equivalent if O has been implemented correctly, but not if O is faulty. Although the technique is only applicable to a small range of fault types, it has been argued that these faults are widespread (McMinn 2009).

Component Harvesting is an alternative procurement strategy. It involves searching online code repositories with some of the desired RI's syntax and semantics specification (Hummel et al. 2006). Hummel et al. (2006) assert that this substantially reduces

[2] Output cardinality refers to the proportion of inputs that map to an output.

procurement effort. However, this may not always be true; other activities may be introduced that will offset effort gains. For example, an RI's relevant functionality may depend on irrelevant functionality; such dependencies must be removed (Janjic et al. 2011). The SUT and RIs must also have a common input-output structure (Chen et al. 2009a). Thus, it may be necessary to standardise the structure of the input and output (Paleari et al. 2010). Atkinson et al. (2011) remark that the effectiveness of the search depends on how well the user specifies the search criteria. It is therefore possible for the search to return systems that cannot be used as RIs. Additionally, systems that have unfavourable legal obligations may also be returned (Atkinson et al. 2011); using these systems as RIs may therefore be infeasible. Identifying and removing such systems from the search results may be labour intensive.

Suitable RIs may not exist (Murphy et al. 2009a). This means Component Harvesting may be inapplicable in some cases. Additionally, the applicability of the technique is restricted by its limitation to simple RIs (Atkinson et al. 2011) i.e. RIs that are limited in terms of scale and functionality. This means that the technique can only support simple SUTs.

Although Testability Transformations and Component Harvesting can substantially improve the usability of N-version Testing, these techniques clearly have limited generalisability i.e. the former and latter only cater for a limited range of faults and systems, respectively. Further research that results in improvements in their generalisability could add significant value. For example, Component Harvesting might be extended to more complex RIs as follows: since the semantics of simple RIs are understood, it may be possible to automatically combine multiple simple RIs into a more complex RI.

3.3 Cost of N-version testing

The cost of N-version Testing is a divisive issue. The cost of obtaining RIs is particularly contentious. Many claim that the RI procurement process is expensive because it may involve the re-implementation of the SUT (Hoffman 1999). However, others argue that this process can be inexpensive because it can be automated by procurement strategies like Component Harvesting (Hummel et al. 2006). However, as discussed in Section 3.2, these strategies are only applicable under certain conditions and so manual re-implementation may be necessary in some situations. This means that the cost of obtaining RIs can vary.

In manual testing, the tester must manually verify the output of a test case. In N-version Testing, this process is automated (Brilliant et al. 1990). This means that test execution can be cheaper in N-version Testing in comparison to manual approaches; thus N-version Testing could be cheaper if a large number of test cases are required. It might be necessary to generate additional test cases because of software maintenance (Assi et al. 2016). Thus, the requirement for a larger number of test cases might be correlated with update frequency. Update cost can be exacerbated by N-version Testing because changes may have to be reflected across all RIs (Lozano 2010). This cost may be further exacerbated, depending on the RI's maintainability (Manolache and Kourie 2001). This may offset the cost effectiveness gains obtained from cheaper test cases in some scenarios.

Let $V1$ be the SUT and $R1$ be an RI based on $V1$. Suppose that $V1$ was updated to become $V2$. Some test cases that are applicable for $V1$ (and by implication, $R1$) may also be applicable for $V2$ (Zhang et al. 2009). Thus, instead of updating $R1$ to be consistent with $V2$, one could simply restrict testing to these test cases. This might alleviate update costs. However, such an approach clearly cannot cater for new functionality (Hummel et al. 2006).

The impact of increasing the number of RIs on cost effectiveness is also unclear. A failure's cost can be high (Janjic et al. 2011; Manolache and Kourie 2001), which means substantial cost savings may be obtained by detecting a fault that could result in such a failure. Since the number of RIs is positively correlated with effectiveness (Brilliant et al. 1990), the chance of obtaining these cost savings can improve if more RIs are used. However, as mentioned above, the cost of developing an RI can be expensive (Lozano 2010). Thus, increasing the number of RIs will inherently increase development cost. Clearly, the direction of the correlation between the number of RIs and cost is dependent on whether a sufficiently expensive fault is found.

It is unclear which system is the source of failure (Chen et al. 2009a); this means that one must debug multiple systems. Thus, using more RIs can lead to an increase in debugging costs. However, using multiple RIs enables the establishment of a voting system, where each RI (and the SUT) votes for its output (Oliveira et al. 2014a). Systems that are outnumbered in a vote are likely to be incorrect. Thus, debugging effort can be directed and therefore minimised. Unfortunately, correct systems can be outnumbered in the vote (Janjic et al. 2011); therefore a voting system may have limited impact in some situations.

3.4 Content-based image retrieval

Some systems produce graphical outputs. The correctness of graphical outputs can be verified by comparing them to reference images (Delamaro et al. 2013). Reference images could be obtained from RIs. Oliveira et al. (2014b) proposed combining a Content-based Image Retrieval System with feature extractors and similarity functions to enable the automated comparison of such outputs with reference images, based on their critical characteristics.

Unfortunately, the application of their technique is not fully automated. For example, one must acquire appropriate feature extractors and similarity functions (Oliveira et al. 2014b). However, some of this manual effort may not always be necessary. For instance, many feature extractors and similarity functions are freely available (Oliveira et al. 2014b), thus one may not have to develop these, if these free ones are appropriate.

4 Metamorphic testing

In Metamorphic Testing (MT), a set of test cases, called the Metamorphic Test Group (MTG), is generated. MTG has two types of test cases. Source test cases are arbitrary and can be generated by any test case generation strategy (Chen et al. 1998b; Guderlei and Mayer 2007b), whilst follow up test cases are generated based on specific source test cases and a Metamorphic Property (Murphy et al. 2008). A Metamorphic Property is an expected relationship between source and follow up test cases.

For example, consider a self-service checkout that allows a customer to scan product barcodes and automatically calculates the total price. The Metamorphic Property might state that a shopping cart that consists of two instances of the same product type should cost more than a shopping cart with just one. Let B_1 and B_2 be instances of the same product, and $SC_1 = \{B_1, B_2\}$ denote a shopping cart containing both instances. Suppose that SC_1 is a source test case. Based on this Metamorphic Property and source test case, MT might use subset selection to derive two follow up test cases: $SC_2 = \{B_1\}$ and $SC_3 = \{B_2\}$. Thus, the MTG may consist of SC_1, SC_2, and SC_3. The Metamorphic Property in conjunction with the MTG is called a Metamorphic Relation (MR). MRs are evaluated by executing the MTG and checking that the Metamorphic Property holds (Kanewala and Bieman 2013b) between these executions; in this case checking that the price of SC_1 is greater than the price of SC_2, and the price of SC_1 is greater than the price of SC_3.

A permutation relation is an MR where changes in the input order has a predictable effect on the output. For example, consider a sort function, $Sort(I)$, where I is a list of integers. A permutation relation might develop the following source and follow up test cases: $Sort(1, 3, 2)$ and $Sort(3, 1, 2)$, with the expectation that their outputs are the same. Some refer to MT as Symmetric Testing in situations where only permutation relations are used (Gotlieb 2003).

Like N-version Testing, MT was created to alleviate the oracle problem. In particular, MT attempts to resolve oracle problems where the test outcome is unpredictable due to a lack of prior knowledge. As has been made apparent above, MT does not rely on predicted test outcomes to verify the correctness of the SUT. Thus, MT can operate in the presence of this oracle problem. MT has also been shown to be effective for a large range of different oracle problems, including complex (Guderlei and Mayer 2007b) (i.e. systems that involve non-trivial processing operations) and data intensive systems (Chen et al. 2009a), because the process of evaluating an MR can be inexpensive.

4.1 Effectiveness of metamorphic testing

Experiments on MT's effectiveness have produced varied results, ranging from 5% (Murphy et al. 2009a) to 100% mutation scores (Segura et al. 2011). Several factors, that influence effectiveness and thus may explain this disparity, have been reported. These factors can broadly be categorised as follows: coverage (Kuo et al. 2010), characteristics, the problem domain, and faults. This section explores these factors. For generalisability purposes, our discussions are limited to implementation independent issues.

4.1.1 Coverage

Numerous strategies for maximising the coverage of MT are available. For example, it has been observed that some MRs place restrictions on source test cases (Kuo et al. 2011). Thus, one's choice of MRs could constrain a test suite's coverage. Coverage could be maximised by limiting the usage of such MRs.

Núñez and Hierons (2015) observed that certain MRs target specific areas of the SUT. This means that increasing the number of MRs that are used, such that the additional MRs focus on areas of the SUT that are not checked by other MRs, could increase coverage.Merkel et al. (2011) state that since testing resources are finite, there is a trade-off between the number of MRs and test cases that can be used. Therefore, increasing the number of MRs to implement the above strategy could limit the test suite size. Thus, the aforementioned coverage gains could be offset. The optimal trade-off is context dependent.

Let P be a program consisting of three paths $P = \{\{s_1, s_2\}, \{s_2\}, \{s_2, s_3\}\}$, and let MR_1 and MR_2 be MRs that each have an MTG that consists of two test cases. Suppose that MR_1's MTG covers the first and second path and thereby executes statements s_1 and s_2, and that MR_2's MTG covers the first and third path and so covers all three statements. This demonstrates that an MR's MTG can obtain greater coverage, if the paths that are traversed by each of its test cases are different (Cao et al. 2013). Several guidelines have been proposed to design MRs to have such MTGs. For example, white-box analysis techniques (Dong et al. 2013), or coverage information generated by regression testing (Cao et al. 2013) could assist in the identification of MRs that have MTGs with different test cases. MRs that use a similar strategy to the SUT tend to have MTGs that have similar source and follow up test cases (Mayer and Guderlei 2006), and thus should be avoided. Different MRs can have different MTG sizes (Cao et al. 2013). It seems intuitive that MRs that have MTGs that consist of a larger number of test cases are more likely to have test cases that traverse dissimilar paths.

4.1.2 Characteristics

An MR has many characteristics that can be manipulated to improve its effectiveness. For example, it has been observed that decreasing the level of abstraction of an MR can improve its fault detection capabilities (Jiang et al. 2014). This section explores these characteristics and their relationships with effectiveness.

MRs can vary in terms of granularity, e.g. application or function level. In a study conducted by Murphy et al. (2013), it can be observed that MRs that are defined at the application level can detect more faults than MRs that are defined at the function level, in some systems. This means that MRs that were defined at a higher level of granularity were more effective for these systems. Interestingly, the converse was also observed for other systems (Murphy et al. 2013), and so the most effective level of granularity might depend on the system. Regardless, both MR types found different faults (Murphy et al. 2013), and thus, both can add value in the same context.

It has been reported that an MR that captures a large amount of the semantics of the SUT (i.e. an MR that reflects the behaviours of the SUT to a greater degree of completeness and accuracy) can be highly effective (Mayer and Guderlei 2006). Let MR_r and MR_p be two MRs, such that MR_r captures more of the semantics of the SUT than MR_p. This suggests that MR_r might be more effective than MR_p. We believe that certain test cases can capture some of the semantics of the SUT. Let tc be such a test case. It may therefore be possible for MR_p to obtain a comparable level of effectiveness to MR_r, if MR_p is evaluated based on tc, because the additional semantics in tc may counteract the deficit of such semantics in MR_p. However, recall that some MRs place restrictions on test inputs (Kuo et al. 2011); this may limit the scope for using test cases like tc with MRs like MR_p.

The fourth widely reported characteristic is strength. Let MR_1 and MR_2 be two MRs, such that MR_1 is theoretically stronger than MR_2. This means that if one can confirm that MR_1 holds with respect to the entire input domain, then this implies that MR_2 also holds with respect to the entire input domain (Mayer and Guderlei 2006). This implies that MR_1 can detect all of the faults that can be detected by MR_2, in addition to other faults. Some regard MRs like MR_2 to be redundant (Sim et al. 2013). Interestingly, a study conducted by Chen et al. (2004a) compared the failure detection rate[3] of 9 MRs. The weakest MR obtained the highest failure detection rate for 15/16 of the faults, whilst the strongest MR obtained the lowest failure detection rate for 13/16 faults. This suggests that strong MRs are not necessarily more effective than weak MRs (Chen et al. 2004a), and weak MRs are therefore not redundant. Mayer and Guderlei (2006) realised that weak MRs can have more failure revealing test cases than stronger MRs. This may explain why weak MRs can be more effective.

Black-box MT emphasises the development of strong MRs. It is therefore not surprising that the observation that weak MRs can be more effective than strong MRs led Chen et al. (2004a) to conclude that black-box MT should be abandoned. Proponents of this argument view an understanding of the algorithm structure as necessary (Chen et al. 2004a). Although this argument has a strong theoretical foundation, Mayer and Guderlei (2006) have questioned the practicality of the position. One must consider all input-output pairs to deduce

[3]The failure detection rate measures the proportion of test cases that detect a fault.

the relative strength of one MR to another, which can be infeasible in practice (Mayer and Guderlei 2006). Thus, opponents contend that categorising MRs based on their strength is impractical, and by implication, deciding to abandon black-box MT based solely on MR strength is nonsensical (Mayer and Guderlei 2006). Whilst we agree with Mayer and Guderlei (2006) that it may be impractical to determine whether one MR is stronger than another, we disagree with the notion that this threatens the validity of the argument of Chen et al. (2004a), since knowledge about the relative strength of two MRs is not necessary to leverage the advice of Chen et al. (2004a).

Tightness is another major characteristic (Liu et al. 2014b); tighter MRs have a more precise definition of correctness. For example, a tight MR may check $X == (Y \times 2)$; only one answer is acceptable. A looser MR may check $(X > 2)$; whilst $(X \leq 2)$ indicates a fault, an infinite number of answers are acceptable. Therefore, tighter MRs are more likely to be effective (Merkel et al. 2011). Although tight MRs are preferable, they may be unavailable. For example, consider a non-deterministic system that returns a random output that is approximately two times larger than the input. A tight MR is not available because predicting the precise output is impossible, however, the following loose MR can be used: $output < (input \times 4)$ (Murphy et al. 2009b).

Another important characteristic is the soundness of an MR. A sound MR is one that is expected to hold for all input values. Conversely an MR that is unsound is only expected to hold for a subset of the input values (Murphy and Kaiser 2010). Unlike sound MRs, unsound MRs are prone to producing false positives[4](Murphy and Kaiser 2010). It might be advisable to avoid using such MRs, to curtail false positives. However, it has been reported that MRs that are less sound might be capable of detecting faults that cannot be detected by MRs that are more sound (Murphy and Kaiser 2010). Thus, such MRs might add value.

4.1.3 Problem domain

It has been reported that MT is more effective when one uses multiple MRs, instead of just one MR (Merkel et al. 2011). Since MRs are domain specific (Chen et al. 2009a), the total number of potential MRs in one problem domain can be different than in another. For example, Chen et al. (2004a) found nine MRs for Dijkstra's Algorithm, whilst Guderlei and Mayer (2007a) could only find one MR for the inverse cumulative distribution function. Therefore, the problem domain is likely to directly influence MT's effectiveness.

Specialised variants of MT have been developed to account for the characteristics of certain problem domains. For example, Murphy et al. (2009a) propose Metamorphic Heuristic Oracles to account for floating point inaccuracies and non-determinism. This approach involves allowing MT to interpret values that are similar, as equal (Murphy and Kaiser 2010). The definition of "similar" is context dependent (Murphy et al. 2009a), thus general guidance is limited.

4.1.4 Faults

MT and its variants can detect a diverse range of faults, e.g. MT can find faults in the configuration parameters (Núñez and Hierons 2015) and specifications (Chen et al. 2009a), and Statistical Metamorphic Testing (see Section 7.3) can find faults that can only be detected by

[4]In the context of software testing, a testing technique is said to have reported a false positive if it incorrectly reports a failure.

inspecting multiple executions (Murphy et al. 2011). However, MRs are necessary, but not sufficient (Chen et al. 2003b); they are not effective for all fault types, e.g. coincidentally correct faults (Cao et al. 2013; Yoo 2010).

Specifications can be used as a source of inspiration for the MR identification process (Jiang et al. 2014; Liu et al. 2014a). It has been reported that the effectiveness of MT can be compromised by errors in the specification (Liu et al. 2014a). This could be because errors in the specification may propagate to the MRs, if the MRs have been designed based on the specification. The same specification errors may have also propagated into the SUT, thus there might be scope for correlated failures (see Section 3.1). This might explain why MT cannot find certain faults. One might reduce this risk by using other sources of inspiration, e.g. domain knowledge (Chen et al. 2009a) or the implementation (Murphy et al. 2008).

4.2 Usability of metamorphic testing

4.2.1 Prerequisite skills and knowledge

Mishra et al. (2013) observed that students performed better on class assignments revolving around equivalence partitions and boundary value analysis, when compared to MT. This suggests that MT might be more difficult to grasp than other testing techniques. This could be because MT requires a wide skillset to operate.

Poon et al. (2014) claim that MR implementation requires limited technical expertise. However, others have stated that the tester might not be competent enough to implement MRs (Chan et al. 2007), which indicates that developing MRs might be difficult. These conflicting conclusions suggest that the difficulty of MR development might be context dependent.

One's domain expectations might not necessarily match the implementation details of the SUT. This disparity might be a result of an intended design decision (Murphy and Kaiser 2008). For example, the SUT's precision may be compromised in favour of efficiency. Thus, if one is not aware of such design decisions and design MRs purely based on domain expectations, the MR might erroneously interpret this disparity as a failure. Thus, knowledge about the implementation details of the SUT might be important.

Domain experts can identify more MRs, that are more effective, more productively than non-domain experts (Chen et al. 2009c). This suggests that domain knowledge is also important. Therefore, if one lacks adequate domain knowledge, it is advisable to consult domain experts (Liu et al. 2012). MRs are identified in booms and slumps; the SUT is investigated during a slump to develop new intuitions that can be used to identify MRs, and such MRs are defined in boom periods (Chen et al. 2016). This iterative process affords further opportunities to continuously supplement one's domain knowledge.

An experiment conducted by Zhang et al. (2009) found that different developers can identify different MRs. This is not surprising because different people have different domain knowledge. It may therefore be advisable to leverage a team (Poon et al. 2014), because this may ensure greater coverage over the domain knowledge. A small team, e.g. consisting of 3 people has been shown to be sufficient (Liu et al. 2014a).

4.2.2 Effort

A number of factors affect the effort required to apply MT. For example, it has been observed that an MR that has been developed for one system, might be reusable in another system (Kuo et al. 2010). Thus, MT might be easier to apply in situations in which MRs that were developed for other systems are available. Another example is MTG size. Since it is not apparent which test case in the MTG reveals the failure, all test cases must be considered during debugging (Chan et al. 2007). This means that effort can be substantially reduced if the MTG size is reduced. Alternatively, Liu et al. (2014b) proposed a method that could provide some indication of the likelihood that a particular test case in MTG executed the fault. Their method deems a test case to be more likely to have executed the fault, if it was executed by more violated MRs. This could be used to direct debugging effort. Another alternative is Semi-Proving, which is covered in Section 4.6.

The most significant factor affecting effort is believed to be the difficulty of MR identification. Thus, most research has been conducted on this factor. For example, Chen et al. (2016) found that MR identification is difficult because inputs and outputs must be considered simultaneously (Chen et al. 2016). They alleviated this by automating input analysis, thereby constraining the tester's attention to outputs (Chen et al. 2016). The technique specifies a set of characteristics, called "Categories"; each is associated with inputs that manipulate it. These inputs are subdivided into "choices"; all inputs belonging to a particular choice manipulate the characteristic in the same way.

A test frame is a set of constraints that define a test case scenario. Pairs of test frames (that correspond to source and follow up test cases) can be automatically generated by grouping various categories and choices together, such that they are "Distinct" and "Relevant" (i.e. marginally different). For example, let $Function(a, b, c, x, y, z)$ be a function with 6 input variables; a distinct and relevant pair may only differ by one of these variables, e.g. z. These test frames produce test cases that are executed to obtain a set of outputs, which can be manually checked for relationships. The process of automatically generating test frames and manually analysing them is iterative (Chen et al. 2016); since an infeasible number of pairs typically exist, the terminating condition is the tester's satisfaction with the identified pool of MRs (Chen et al. 2016).

The empirical evidence is promising; people's performance with respect to MR identification improved, and novices achieved a comparable level of performance to experts (Chen et al. 2016). However, the technique has an important limitation; it can currently only support MRs that are composed of one source and follow up test case (Chen et al. 2016).

Kanewala and Bieman (2013b) alternatively propose training Machine Learning classifiers to recognise operation sequence patterns that are correlated with particular MRs. Such a classifier can predict whether unseen code exhibits a particular MR. Results have been promising; the technique has a low false positive rate, and can identify MRs even when faults are present in the SUT (Kanewala and Bieman 2013b).

Although this technique achieves greater automation (Kanewala and Bieman 2013b) than the approach devised by Chen et al. (2016), additional human involvement is introduced elsewhere. For example, training datasets are necessary for the machine learning classifiers (Kanewala et al. 2014), and obtaining these can be difficult (Chen et al. 2016). One may wish to extend the classifier with a graph kernel,[5] that has parameters (Kanewala et al. 2014) that might have to be tuned to improve accuracy. Furthermore, since each classifier

[5]A graph kernel is a function that compares graphs based on their substructures.

is associated with one MR type (Kanewala and Bieman 2013b), these additional tasks must be repeated for each MR type.

4.3 Efficiency of metamorphic testing

There is a time cost associated with test case generation and execution (Chen et al. 2014c). As discussed in Section 4.1.1, different MRs have different MTG sizes. This means that some MRs might incur greater time costs than others. Thus, one might improve the efficiency of MT by restricting oneself to MRs with smaller MTGs. However, as was discussed in Section 4.1.1, MRs with larger MTGs might obtain greater coverage, thus such a restriction might reduce the effectiveness of the technique. Alternatively, one might consider using parallel processing — the test cases in the MTG can be executed simultaneously (Murphy and Kaiser 2010).

Other approaches include combining MRs in various ways to make more efficient use of test cases. For example, one could use the same test cases for different MRs (Chen et al. 2014c). One method of implementing such an approach might be Iterative Metamorphic Testing. This involves combining MRs together, $\langle MR_i, MR_{i+1}, MR_{i+2}...MR_n\rangle$, such that the follow up test case(s) of MR_i are used as the source test case(s) of MR_{i+1} (Wu 2005). Combination relations is another possible method.

4.4 Combination relations

Liu et al. (2012) suggested defining a new MR that is composed of multiple MRs. For ease of reference, we called such an MR a "combination relation". By evaluating this single MR, one implicitly evaluates all of the constituent MRs, and thus makes more efficient use of test cases. Logic dictates that a single MR that embodies multiple MRs would have a level of effectiveness that is equivalent to the sum of its constituent parts (Liu et al. 2012). Interestingly however, it has been found that such an MR can actually obtain a higher level of effectiveness than its constituent MRs (Liu et al. 2012). This could be because one MR in the combination relation may empower another. For example, MRs MR_n and MR_c may be effective for numerical and control flow faults, respectively; combining the two may extend MR_n's capability to control flow faults.

Conversely, effectiveness can deteriorate; Liu et al. (2012) observed that including a loose MR in a combination relation can reduce the combination relation's overall effectiveness. Thus, they advocate only combining MRs that have similar tightness. They also observed that some MRs might "cancel" out other MRs either partially or completely (Liu et al. 2012). This may also explain why the effectiveness of a combination relation might deteriorate. These observations suggest that one may be limited in one's choice regarding which MRs can be combined, and by implication, the technique might be inapplicable in some scenarios (Monisha and Chamundeswari 2013).

Different MRs can accommodate different subsets of the input domain (Dong et al. 2007). Since an input must be suitable for all MRs in the combination relation, additional restrictions might have to be placed on constituent MRs. For example, suppose that MR_1 and MR_2 are two MRs in a combination relation. MR_1 can accommodate five test inputs, $\{t_a, t_b, t_c, t_d, t_e\}$, and MR_2 can only accommodate three test inputs, $\{t_a, t_b, t_f\}$. In this situation, it is not possible to use test cases $\{t_c, t_d, t_e, t_f\}$. This means that MRs that can accommodate larger subsets may be more useful (Dong et al. 2007). This could also explain why a combination relation's effectiveness might deteriorate.

4.5 Metamorphic runtime checking

In Metamorphic Runtime Checking, MRs are instrumented in the SUT, and evaluated during the SUT's execution. One of the benefits of this approach is that MRs are evaluated in the context of the entire SUT (Murphy et al. 2013). This can improve the effectiveness of MT. To illustrate, Murphy et al. (2013) observed that MRs that are evaluated in one area of the system, could detect faults in other areas.

Unfortunately, unintended side effects can be introduced during instrumentation (Murphy et al. 2013). For example, consider a function, $F(x)$, and a global counter variable, I. I is incremented every time $F(x)$ is executed. A follow up test case that executes $F(x)$ will inadvertently affect I's state. Thus, sandboxing may be advisable (Murphy and Kaiser 2010).

Sandboxes introduce additional performance overheads (Murphy and Kaiser 2009). However, since Metamorphic Runtime Checking uses test data from the live system (Murphy et al. 2009b), the generation of a source test case is no longer necessary. These efficiency gains may offset the losses from the performance overheads incurred from sandboxes.

To improve the efficiency of the approach further, some have suggested parallel execution (Murphy and Kaiser 2010). It has been observed that the number of times each MR is evaluated is dependent on the number of times each function is invoked (Murphy et al. 2013). To illustrate, let $f_1()$ and $f_2()$ be two functions in the same system, such that $f_1()$ is always invoked twice as many times as $f_2()$ because of the control flow of the system. Suppose that MRs MR_1 and MR_2 are evaluated each time $f_1()$ and $f_2()$ are invoked, respectively. Since MR_1 is evaluated twice as many times as MR_2, MR_1 would add more performance overheads than MR_2. Thus, one could prioritise MRs like MR_2 over MRs like MR_1 to improve performance.

4.6 Semi-proving

An MR's verdict only indicates the SUT's correctness for one input. Semi-Proving attempts to use symbolic execution to enable such a verdict to generalise to all inputs (Chen et al. 2011b).

In Semi-Proving, each member of an MR's MTG, $MetTestGrp = \{tc_1, tc_2, ...tc_n\}$, is expressed, using symbolic inputs, as constraints that represent multiple concrete test cases. Each test case, tc_i, in MTG is symbolically executed, resulting in a set of symbolic outputs, $O_i = \{o_{ij}, o_{ij+1}, ...o_{in}\}$, and corresponding symbolic constraints, $C_i = \{c_{ij}, c_{ij+1}, ...c_{in}\}$, that the output is predicated on. Let CP be the Cartesian product of each C_i i.e. $C_1 \prod C_2 \prod ... \prod C_n$. For each combination $comb = \langle C_{1a}, C_{2b}, ...C_{nc}\rangle$ in CP, the conjunction of all members of $comb$ should either result in a contradiction or agreement. For each agreement, Semi-Proving checks whether the MR is satisfied or violated under the conditions represented by $comb$.

Since all concrete executions represented by a symbolic execution are accounted for, it is possible to prove the correctness for the entire input domain, with respect to a certain property (Chen et al. 2011b). However, this might not always be feasible. For example, in some systems, certain loops, arrays, or pointers could cause such a large number of potential paths to exist, that it would be infeasible for Semi-Proving to check them all exhaustively (Chen et al. 2011b). To alleviate this problem, one could restrict the application of the technique to specific program paths, replace some symbolic inputs with concrete values, use summaries of some of the SUT's functions instead of the functions themselves, or restrict the technique with upper-bounds (Chen et al. 2011b). Chen et al. (2011b) realised that the correctness of some symbolic test cases can be inferred from others. For example, consider the max function and the following two symbolic test cases: $max(x, y)$ and $max(y, x)$. Since these test cases are equivalent, only one must be executed to deduce the correctness of both. Optimising resource utilisation through this strategy may also alleviate the problem.

Obtaining such coverage can improve the fault detection effectiveness of MT (Chen et al. 2011b). Improvements in effectiveness for subtle faults, e.g. missing path faults, has been reported to be particularly noteworthy by several researchers (Chen et al. 2011b; Gotlieb and Botella 2003). Another advantage of greater coverage is improvements in debugging information. In particular, there is greater scope for the precise failure causing conditions (Chen et al. 2011b) and test cases (Liu et al. 2014b) to be identified. Whether this improves debugging productivity is questionable though; investigating this information requires manual inspection of multiple (possibly all) execution paths (Chen et al. 2011b).

4.7 Heuristic test oracles

Heuristic Test Oracles are a loose variant of Metamorphic Testing. In this approach, expected input-output relationships are initially identified, e.g. input increase implies output decrease. The SUT is then executed multiple times with different inputs, to obtain a set of outputs. These inputs and outputs are used in conjunction with each other to check whether the expected input-output relationship holds (Hoffman 1999).

Thus, Heuristic Test Oracles can only be applied to systems that have predictable relationships between inputs and outputs (Hoffman 1999). Some systems may not have relationships that span the entire input domain. In such situations, it might be possible to define heuristics for a subset of the input domain (Hoffman 1999). For example, Sine's input domain can be split into three subdomains: $Subdomain\ One = \{0 \leq i \leq 90\}$, $Subdomain\ Two = \{90 \leq i \leq 270\}$, and $Subdomain\ Three = \{270 \leq i \leq 360\}$. A positive correlation between the input and output can be observed in $Subdomain\ One$ and $Subdomain\ Three$, whilst a negative correlation is assumed in $Subdomain\ Two$ (Hoffman 1999).

It has been reported that these oracles are effective (Lozano 2010). Some have also claimed that these oracles are faster and easier to develop (Hoffman 1999) and maintain (Lozano 2010) than N-version Testing based oracles. Heuristic Test Oracles also have high reuse potential (Hoffman 1999), thus implementation may be bypassed completely in some cases.

5 Assertions

Assertions are Boolean expressions that are directly embedded into the SUT's source code (Baresi and Young 2001). These Boolean expressions are based on the SUT's state variables, e.g. $X > 5$, where X is a state variable. Assertions are checked during the execution of the SUT, and may either evaluate to true or false; false indicates that the SUT is faulty (Harman et al. 2013). Our general discussions on Assertions in this section are based on the above definition. We are aware that some people use alternative definitions. For example, some definitions allow one to augment the SUT, e.g. by introducing auxiliary variables (see Section 5.1), and other definitions consider runtime exceptions to be "free" assertions. Our discussions regarding such alternative definitions of the technique will be clearly indicated in the text.

Unlike N-version Testing and Metamorphic Testing, Assertions were not originally designed to alleviate the oracle problem. However, it has been observed that in order to evaluate an assertion, one does not have to predict the test outcome (Baresi and Young 2001). This means that assertions are applicable to certain classes of oracle problem, e.g. for situations in which it is not possible to predict the test outcome.

5.1 Effectiveness of assertions

Several characteristics of Assertions have been found to influence effectiveness. For example, one characteristic is that Assertions must be embedded in source code (Sim et al. 2014). Unfortunately, this can cause unintended side effects that manifest false positives, e.g. additional overheads (Kanewala and Bieman 2014) may cause premature time-outs. Thus, one must carefully write assertions to avoid side effects (Murphy and Kaiser 2009).

Assertions can be written in independent programming or specification languages, e.g. assertions can be written in Anna, and be instrumented in a program written in Ada (Baresi and Young 2001). Some languages are particularly intuitive for certain tasks, e.g. LISP for list manipulation. One could exploit these observations, by writing Assertions in the most apposite language for the types of tasks to be performed. This might reduce the chance of introducing unintended side effects. Unfortunately, this approach can also increase the chance of introducing unintended side effects if it causes deterioration in readability. One can use polymorphism; assertions can be specified in a parent class, and a child class can inherit assertions from the parent class (Araujo et al. 2014). By using such a strategy, one can isolate assertions (in parent classes) from the system's source code (in child classes); this might alleviate readability issues.

The code coverage of Assertions can be limited, depending on the nature of the program. For example, let $List$ be an array. To test $List$, an Assertion may assert that some property holds for all members of $List$. It may be infeasible to evaluate this Assertion, if $List$ has an large number of elements (Baresi and Young 2001). Thus, it may be infeasible for assertions to be used in areas of the code that have large arrays. Consider another example; Assertions can only check a limited range of properties that are expected to hold at particular points in the program, e.g. $Age \geq 0$ (Harman et al. 2013). This means their coverage could be limited. According to some alternative definitions of Assertions, auxiliary variables can be introduced into the system, for the purpose of defining Assertions (Baresi and Young 2001). Introducing auxiliary variables might create new properties that can be checked by Assertions, and thus alleviate the problem. For example, suppose that x is a variable in the system, and y is a newly introduced auxiliary variable; we might include an assertion such as $x > y$.

Another facet of coverage is oracle information. One aspect of oracle information is the types of the properties that can be checked by a technique. For example, Assertions can check the characteristics of the output or a variable, e.g. range checks (Sim et al. 2014), or how variables might be related to one another (Kanewala and Bieman 2013a), e.g. $X \neq Y$. This makes Assertions particularly effective for faults that compromise the integrity of data that is assigned to variables (Murphy and Kaiser 2010). Another aspect of oracle information is the number of executions that test data is drawn from. Test data from multiple executions is necessary for certain faults, e.g. the output distributions of a probabilistic algorithm (Murphy et al. 2011). Assertions are unable to detect such faults because they are restricted to one execution (Murphy et al. 2011).

Some believe that Assertions can be used to detect coincidentally correct faults (Kanewala and Bieman 2013a). This supposition probably stems from the fact that Assertions have access to internal state information, and thus could detect failures in internal states, that do not propagate to the output. To the best of our knowledge, there isn't any significant empirical evidence that demonstrates that Assertions can cope with coincidental correctness. Thus, investigating this might be a useful future research direction.

It has been observed that the detection of some coincidentally correct faults may require oracle information from multiple states (Patel and Hierons 2015). Based on an alternative definition of Assertions, Baresi and Young (2001) report that Assertions can check multiple states, if they are used in conjunction with state caching. However, they also remark that state caching may be infeasible, if a large amount of data must be cached. In such situations, Assertions cannot detect such coincidentally correct faults. Additionally, they observed that Assertions cannot correlate events between two modules that do not share a direct interface. This means that assertions may not be able to check certain states simultaneously, and thus may render it incapable of detecting certain coincidentally correct faults. These observations demonstrate that, despite the fact that assertions have access to internal state information, they may not necessarily be effective for coincidental correctness, even when state caching is feasible.

MT has access to information from multiple executions. Sim et al. (2014) combined MT and Assertions, such that Assertions are evaluated during the execution of a Metamorphic Test's source and follow up test cases. This integration may alleviate some of the oracle information coverage issues described above.

5.2 Usability of assertions

One key skill that is a part of many developers repertoires is program comprehension i.e. the capability to understand the logic of a program by inspecting the source code (Zhang et al. 2009). Developers have experience with modifying source code (Zhang et al. 2009), e.g. to add new functionality. Therefore, developers will be comfortable with comprehending and modifying the system's source code. These tasks are integral to the application of assertions. This led Zhang et al. (2009) to conclude that constructing assertions can be more natural than developing oracles from other approaches like Metamorphic Testing. However, Assertions assumes that the tester has knowledge about the problem domain, or the SUT's implementation details (Kanewala and Bieman 2013a). This means that an assertion could require more effort to construct in situations in which the tester has limited knowledge regarding these areas, since they would have to first acquire this knowledge. Other factors that affect the effort required to construct an assertion include the level of detail the assertion is specified at Araujo et al. (2014) and the programming language's expressiveness (Nardi and Delamaro 2011).

Some tools can support the development of assertions, e.g. the assert keyword in some programming languages (Baresi and Young 2001), and invariant detection tools. Invariant detection tools can be used to automatically generate assertions. They work by conducting multiple executions and recording consistent conditions (Kanewala and Bieman 2013a); these conditions are assumed to be invariant and so pertain to assertions. It is typically infeasible to consider all executions; thus only a subset is used. Variant conditions may be consistent across this subset and thus may be misinterpreted as invariant. Thus, invariant detection tools can produce spurious assertions (Murphy et al. 2013). Therefore, the manual inspection of suggestions from these tools is necessary (Kanewala and Bieman 2013a). Unfortunately, manual inspection can be error prone; cases have been observed where 50% of the incorrect invariants that were proposed by such a tool were misclassified by the manual inspection process (Harman et al. 2013).

5.3 Multithreaded programs

Interference in multi-threaded environments can cause assertions to produce false positives (Araujo et al. 2014). Several guidelines have been proposed to circumvent this. Firstly, assertions can be configured to evaluate under safe conditions, e.g. when access to all required data has been locked by the thread (Araujo et al. 2014). Secondly, the application of assertions can be restricted to blocks of code that are free from interference (Araujo et al. 2014). Recall that assertions can add performance overheads. This is problematic in multi-threaded environments, because these performance overheads can introduce new or remove important interleavings. This can be alleviated by load balancing (Araujo et al. 2014).

5.4 Further discussion

Research on assertions in the context of the oracle problem is scarce. Most studies either combine it with other techniques or use it as a benchmark. This implies that Assertions are assumed to be at least moderately effective for non-testable programs; but this is largely unsubstantiated. Thus, empirical studies that test this assumption may be valuable.

The literature reported in this Mapping Study did not present guidelines for assertion use in non-testable systems. We therefore believe that future work that establishes such guidelines in the context of the oracle problem will be valuable.

6 Machine Learning

Machine Learning (ML) Oracle approaches leverage ML algorithms, in different ways, for testing purposes. One method involves training a machine learning algorithm, on a training dataset, to identify patterns that are correlated with failure. The SUT can be executed with a test case, and this trained machine learning algorithm can then be used to check for such patterns in this test case execution. For example, Chan et al. (2009) constructed a training dataset, in which each data item corresponded to an individual test case, and consisted of a set of features that characterised the input and output of this test case. Each data item was also marked as "passed" or "failed". A classifier was trained on this training dataset and so became capable of classifying test cases that were executed by the SUT, as either passed or failed. Another method involves training a machine learning algorithm to be a model of the SUT; thus, the ML algorithm becomes akin to a reference implementation in N-version Testing (Oliveira et al. 2014a).

ML techniques were not originally developed for testing non-testable programs, but they can be applied to such programs (Kanewala and Bieman 2013a). To illustrate, ML Oracles draw their oracle information from training datasets, which can be obtained when information about the expected test outcome is not available prior execution. This can allow them to test systems for which the expected test outcome is not known before the execution.

6.1 Effectiveness of machine learning oracles

6.1.1 Design and application of machine learning oracles

Several factors affect the effectiveness of ML Oracles. The first set of factors concerns the composition of the training dataset. It has been reported that the balance of passed and failed test cases can affect bias (Chan et al. 2009). Datasets can also vary in terms of size. Larger datasets have less bias (Chan et al. 2009) and are less susceptible to the negative effects of noise (Frounchi et al. 2011).

A training dataset must often be reduced to a set of features that characterise it, because the form of the training dataset is seldom appropriate for ML. This is typically achieved by using one or more feature extractors. The second set of factors revolves around the number of feature extractors one uses. Several trends between the number of feature extractors used and effectiveness can be observed: improvement, stagnation, and decline. Two of the feature extractors used in a study conducted by Frounchi et al. (2011) include the Tanimoto Coefficient TC and Scalable ODI $SODI$. In this study, it was observed that one set of features that consisted of $\{TC\}$ was the most effective set for negative classifications, and that another set of feature extractors that contained $\{TC, SODI\}$ was the most effective set for positive classifications. Clearly, the addition of $SODI$ to a set of feature extractors that just contains TC can lead to an increase in the accuracy for one type of classification, but a decrease for another type of classification. This implies that an ML Oracle's overall effectiveness can be improved or reduced by adding additional feature extractors, if the improvement in one classification type more than offsets, or is more than offset by the loss of accuracy for other classification types, respectively. These implications might explain why one may observe the improvement and decline trends.

Let $G = \{fe_1, fe_2, ...fe_j\}$ be a group of feature extractors, such that all $fe_i \in G$ are highly correlated with one another. Using multiple members from G is unlikely to significantly improve classification accuracy (Frounchi et al. 2011). This could explain stagnation trends. This suggests that one should limit the number of members of G, that are used by ML Oracles. Different feature extractors may have different quality attributes, e.g. levels of efficiency (Frounchi et al. 2011) or generalisability and so some may be more favourable than others in certain contexts. Thus, one may consider choosing a subset of G based on the quality attributes offered by the different feature extractors in G. Techniques like wrappers and filters can identify and remove feature extractors that will not significantly improve classification accuracy (Frounchi et al. 2011) and thus can purge excess members of such a group.

Naturally, one would expect that one major factor that might affect effectiveness, is the choice of ML algorithm. However, it has been reported that the choice of algorithm does not have a significant impact on effectiveness, and thus, these algorithms might be interchangeable (Frounchi et al. 2011).

6.1.2 Limitations

ML Oracles have several limitations, and to the best of our knowledge, these limitations have not been resolved by the community yet. Recall that ML Oracles either predict the output of the SUT and then compare this prediction to the SUT's output, or they classify the output of the SUT as correct or incorrect. This means that such oracles are fundamentally used for black-box testing. It's therefore not surprising that examples of these oracles cannot test event flow (Nardi and Delamaro 2011). For similar reasons, such oracles would be hindered by coincidental correctness. Some have also observed that the negative impact of coincidental correctness on ML Oracles can be exacerbated, if the ML Oracle is trained based on features of the internal program structure (Kanewala and Bieman 2013a; Chan et al. 2010). It has also been reported that some variants of ML Oracles are incapable of testing non-deterministic systems or streams of events, e.g. ML oracles based on Neural Networks (Nardi and Delamaro 2011). We believe that resolving these limitations might be useful avenues for future work.

6.2 Usability of machine learning oracles

6.2.1 Design and application of machine learning oracles

The previous section revealed that in order to leverage ML Oracles, one must obtain appropriate training datasets, an ML algorithm, and apposite feature extractors. This section explores the user-friendliness of these activities.

We begin by considering training dataset procurement. One approach might include obtaining an RI of the SUT and then generating the training dataset from this RI (Chan et al. 2006). RIs have several characteristics that influence dataset quality, e.g. the correctness of the RI. To illustrate, an RI might have a fault, which means that some of the training samples in the dataset may characterise incorrect behaviours (i.e. failures that manifested from this fault), but be marked as correct behaviours. This reduction in dataset quality can limit the effectiveness of an ML Oracle. For example, the SUT might have the same fault as the RI (Kanewala and Bieman 2013a), and this can lead to correlated failures. In addition, it has been observed that the extent to which an RI is similar to the SUT is correlated with accuracy (Chan et al. 2009), and that oracles based on similar RIs can be accurate, effective, and robust (Kanewala and Bieman 2013a). These discussions reveal that one must consider a large number of factors during the RI selection process, which could be difficult.

The nature of the data in the training dataset could also have an impact on effort. As discussed above, one aspect of a dataset's composition is test suite balance (in terms of the proportion of passed to failed test cases). If one's dataset is imbalanced, it may be necessary to expend additional effort to obtain additional data to supplement and balance the dataset. The output of an RI characterises correct behaviours, and the output of mutants of an RI characterise incorrect behaviours (Chan et al. 2009). Thus, if one lacks passed test cases, one could execute an RI with test cases, and if one lacks failed test cases, one could execute failure revealing test cases over a set of mutants of an RI. However, one may have to construct raw datasets manually, if a suitable RI does not exist (see Section 3.2).

The nature of the input and output data used and produced by an ML algorithm can differ from that of the SUT. Thus, it could be necessary to translate inputs that are used by the SUT into a form that is compatible with the ML algorithm and to translate outputs into a form that is amenable for comparison with the SUT's output (Pezzè and Zhang 2014). If such translations are necessary, the developers of ML Oracles must either write additional programs to automate this translation task, or perform the translation task manually.

Experts may have to manually label each training sample in the dataset, if one uses a supervised machine learning algorithm to train one's ML Oracle. An example of this can be found in the work conducted by Frounchi et al. (2011). This obviously means that larger datasets will require substantially more effort to prepare, in these situations. If multiple experts are used, then there is scope for disagreement (Frounchi et al. 2011). The resolution of these disagreements will also add to the overall effort required to apply the technique.

We finally consider feature extractor selection. One's choice of feature extractors is an important determinant of the effectiveness of ML Oracles. For this reason, many believe that a domain expert should be involved in this process (Kanewala and Bieman 2013a). If the developer is not a domain expert, consultation may be necessary.

6.2.2 Debugging

ML Oracles can report false positives (Chan et al. 2009), which means testers may waste time investigating phantom faults. ML Oracles can also produce false negatives (Kanewala and Bieman 2013a). False negatives introduce a delay, which means they can also waste resources. Some ML Oracles have tuneable thresholds. Modifying these thresholds can influence the incidence of false positives and negatives (Nardi and Delamaro 2011), which can enable management of such classification errors. Unfortunately, the optimal threshold values vary across systems (Nardi and Delamaro 2011).

6.3 Metamorphic machine learning

Metamorphic Machine Learning merges MT and ML, such that an ML Oracle evaluates each member of the MTG, before they are used to evaluate the MR. The integration of MT with ML has been found to improve the effectiveness of ML (Chan et al. 2010). However, the level of this improvement depends on the quality of the ML Oracle. To illustrate, Chan et al. (2010) observed that the extent of the improvement for ML Oracles that used more training data (and were therefore of higher quality) was lower. They rationalised that this was because there was less scope for MT to offer an improvement. Since ML can detect a fault before all of the test cases in the MTG have been executed (Chan et al. 2010), one could argue that the union of MT and ML can also enhance the efficiency of MT, because it may not be necessary to execute all test cases to detect a fault.

7 Statistical hypothesis testing

In Statistical Hypothesis Testing (SHT), the SUT is executed multiple times to obtain numerous outputs, which are aggregated using summary statistics, e.g. mean and variance. These aggregated values characterise the distribution of this set of outputs and are compared (using a statistical test, e.g. Mann-Whitney U) to values that delineate the expected distribution. Comparisons that do not yield significant differences can be interpreted as evidence that the SUT behaved correctly (Ševčíková et al. 2006), and significant differences are evidence of the contrary.

The test outcome of a system can be unpredictable because of non-determinism, which means that such systems are instances of the oracle problem. SHT was developed to resolve this specific type of oracle problem (Guderlei and Mayer 2007a). SHT recognises that such systems may have a typical output distribution, and that information about this typical output distribution may be available prior to execution, even if information about the test outcome of a single execution is not. Since it conducts testing by checking the SUT's output distribution against the typical output distribution, it can be applied in situations where it is not possible to predict the test outcome of a single execution.

7.1 Assumptions

The generalisability of SHT is limited (Sim et al. 2013), because the technique makes several assumptions that may not always hold. For example, the SUT or input generation method must be non-deterministic (Mayer and Guderlei 2004). Thus, the technique is not applicable to scenarios in which the SUT is deterministic, and random testing is not used. Another example of such an assumption is that the expected output distribution is known (Mayer 2005). One could use reference implementations (RIs) to determine the expected distribution (Guderlei and Mayer 2007a), if this assumption does not hold. Unfortunately, the negative issues that are associated with the use of RIs may also affect SHT, if this approach is used, e.g. correlated failures. Another issue could be that an RI may not be available (Guderlei and Mayer 2007a), thus the technique might not always be applicable.

The statistical techniques used in SHT make assumptions about the data. This means that some statistics may not be applicable to certain data samples that are produced by a system because these data samples may not satisfy the assumptions of these statistics. To illustrate, Ševčíková et al. (2006) investigated a simulation package and found that the data that was produced by this system either adhered to Normal or Poisson distributions. Parametric statistics assume that the distribution is Normal and so may not be applicable to all of the data samples produced by their simulation package.

In situations in which a test statistic's assumptions have been broken, one could use a different statistic that does not make such an assumption. For example, one could use a non-parametric statistic, if the data is abnormally distributed. However, it has been reported that non-parametric statistics are less effective (Guderlei et al. 2007), thus doing so may compromise the effectiveness of SHT. Alternatively, it might be possible to modify data samples to satisfy the broken assumptions. For example, Ševčíková et al. (2006) used a test statistic that assumed that variance was constant across all dimensions of an output, but remarked that such an assumption may not always hold. They also stated that log or square root transformations could be used to stabilise the variance (Ševčíková et al. 2006). Thus, performing such transformations may resolve the issue.

7.2 Effectiveness of statistical hypothesis testing

Ševčíková et al. (2006) compared the performance of Pearson's χ^2 with a statistic they called $LRTS_{poisson}$ and found that the latter was more powerful. This suggests that the effectiveness of SHT is partly dependent on the choice of statistical test, and that one should always opt to use the most effective, applicable statistical tests.

The summary statistics that characterise the distributions are also an important determinant of effectiveness. To illustrate, Guderlei et al. (2007) found that variance was more effective than mean. Interestingly, they also observed that the variance and mean detected mutants that the other failed to detect. This indicates that one should use multiple summary statistics.

SHT's performance was abysmal in an experiment conducted by Guderlei et al. (2007). In this experiment, SHT only considered characteristics of the SUT's output, instead of the entire output. The authors suspect that this explains SHT's performance. This suggests that one should maximise the amount of data being considered by SHT to enhance its effectiveness. However, one of the findings of an experiment conducted by Ševčíková et al. (2006) was that tests that considered fewer dimensions of the output could be more effective. This indicates the converse i.e. reducing some of the data being considered by SHT could improve effectiveness. These conflicting observations suggest that the most appropriate amount of data to expose SHT to is context dependent. We believe that future work that establishes a set of guidelines with respect to the most apposite amount of data to make available to SHT would be valuable.

Yoo (2010) exposed a variant of SHT, called Statistical Metamorphic Testing (see Section 7.3), to different datasets. He observed that the dataset that offered the worst performance may have had outliers and suggested that this may explain its comparatively poorer performance to other datasets. This suggests that the nature of the data is also important.

SHT is necessary, but not sufficient; false positives and negatives are possible (Ševčíková et al. 2006). In SHT, one has control over the significance level. Higher significance levels result in more false positives, but fewer false negatives (Ševčíková et al. 2006) and vice versa. Thus, one can tune the significance level to enable better management of these classification errors.

It is unclear which test cases are incorrect (Mayer 2005); thus manual inspection of each is necessary. This suggests that reducing the size of the sample might be beneficial from a debugging perspective. However, it has also been observed that increasing the sample size can lead to an increase in the number of faults that can be detected by the technique (Guderlei et al. 2007). Thus, reducing the size of the test suite could lead to a reduction in effectiveness. Unsurprisingly, it has been reported that SHT can be very resource intensive because it requires a large number of executions to produce stable results (Guderlei et al. 2007). This means reducing the test suite size might also be beneficial from an efficiency viewpoint, but doing so may compromise the stability of the technique. There are clearly several trade-offs associated with the sample size that might affect the effectiveness of the technique.

7.3 Statistical metamorphic testing

Recall that SHT assumes that one either has knowledge about the expected output distribution, or a reference implementation that can determine the expected distribution. Guderlei and Mayer (2007a) combined SHT with MT to ameliorate this assumption. The integrated approach is called Statistical Metamorphic Testing. The approach operates as follows. For a given MR, the source and follow up test cases are executed multiple times to obtain two or more sets of outputs. Each set is aggregated into one statistical value, and a statistical hypothesis test is evaluated based on these values. This integrated approach also enhances MT's capability to operate in non-deterministic systems (Yoo 2010).

The integration of these techniques can clearly be advantageous in some respects, e.g. from a generalisability perspective. However, the union of these techniques can also be detrimental in other ways. For example, it was reported that in Statistical Metamorphic Testing, the most appropriate statistical analysis is dependent on the MR (Yoo 2010). This means one must expend additional effort to determine the most appropriate statistical analysis for each MR, which is an otherwise unnecessary task in standard MT.

Yoo (2010) investigated the effectiveness of Statistical Metamorphic Testing and found that it is affected by choice of statistical hypothesis test and choice of test cases. He also noted that Statistical Metamorphic Testing was incapable of detecting faults that failed to propagate to the output i.e. cases of coincidental correctness.

8 Comparing techniques

Sections 3 to 7 described a series of techniques that were devised to alleviate the oracle problem. Each technique was explored in terms of its effectiveness, efficiency, and usability. Sections 8.1, 8.2, and 8.3 compare these techniques on the basis of these issues.

8.1 Effectiveness

Certain faults can only be detected by assessing specific oracle information, e.g. specific test cases may be necessary to detect certain faults. Table 5 shows that different techniques have access to different oracle information and thus may find different faults. For example, since Assertions only evaluates the SUT based on oracle information from a single execution,

Table 5 A summary of the effectiveness data for each technique, based on Sections 3 to 7

Technique	Can this technique experience correlated failures?	Is this technique effective for coincidental correctness?	Examples of choices that one can make with respect to the design and application of an oracle based on this technique.	Examples of ways in which the coverage of this technique is restricted, in terms of oracle information.
N-Version Testing	True	False	One can choose whether the programming language that an RI is written in, is the same as the programming language of the SUT.	N-version Testing is a black-box testing technique, and so it cannot use white-box oracle information.
Metamorphic Testing	True	False	One can choose the tightness of an MR.	Some MRs place restrictions on source test cases. This means the code coverage of such MRs might also be restricted.
Assertions	Unknown	Unknown	One can decide whether or not to introduce Auxiliary Variables.	Assertions are limited to checking a single execution. Thus, Assertions cannot draw oracle information from multiple executions, simultaneously.
Machine Learning Oracles	True	False	One has a choice over which feature extractors will be used by the technique.	This is a black-box testing technique and so is restricted to verifying the correctness of the SUT's output.
Statistical Hypothesis Testing	True	False	One has a choice over which statistical tests will be conducted by the technique.	If the SUT is deterministic, then Statistical Hypothesis Testing mandates that one uses random testing, as the test case generation strategy.

it cannot detect faults that require oracle information from multiple executions. Statistical Metamorphic Testing has access to such information and so can detect such faults. However, some MRs place restrictions on which test cases can be used. Let TCF be the set of test cases that can manifest a particular fault, F. If an MR's restrictions prevent it from using members of TCF, then it will not be able to detect F. Assertions do not have this restriction and so might be able to detect F. Clearly, practitioners should select testing techniques based on the types of faults that their system is prone to. This highlights some research opportunities; in particular, it may be possible to extend the types of faults that one technique can detect, by combining it with another technique that uses different oracle information. An example of this was presented at the end of Section 5.1.

Although different techniques can find different types of faults, they might not be able to detect all instances of these faults. Every technique has limitations in terms of coverage (see Table 5), thus this potential explanation applies to all of the techniques. Alternatively, correlated failures may explain this phenomenon for a subset of the techniques (see Table 5). A large amount of research has been conducted on reducing correlated failures for N-version Testing, but very little has been done in the context of other techniques that are known to experience correlated failures. We therefore believe that such research could be a valuable asset to the community.

Table 5 outlines an example of a design and application option for each technique. Sections 3 to 7 revealed that some techniques have more design and application options than others. Such techniques offer a greater degree of control; this might enable better optimisation of the technique for different contexts. However, it may be more difficult to find a suitable design and mode of application for such techniques.

One's choices regarding a technique's design and application options can have both a positive and negative impact. For example, increasing the number of feature extractors used by an ML Oracle can lead to improvements in one type of classification, but reductions in another. Unfortunately, guidelines on how one should exploit many of these types of design and application options for their context have not been proposed. Research that leads to the establishment of such guidelines would be useful.

Unfortunately, to the best of our knowledge, empirical data regarding the effectiveness of Assertions for coincidental correctness is unavailable. We therefore believe that significant value can be gained by studying this technique in the context of coincidental correctness and the oracle problem. Interestingly, Sections 3 to 7 suggest that the remaining techniques can be ineffective for coincidental correctness (see Table 5). Thus, research that explores methods of reducing the impact of coincidental correctness on these techniques would be valuable. For example, Clark and Hierons (2012) and Androutsopoulos et al. (2014) developed a series of metrics that estimate the probability of encountering coincidental correctness on particular program paths. Such metrics can be used to select test cases that are less susceptible to coincidental correctness.

8.2 Efficiency

Table 6 reveals that the contexts in which the different techniques perform particularly poorly may differ. For example, the feature extractors that are available in a certain context may be particularly inefficient, but the SUT in this context may have very few large arrays. This means that assertions and machine learning may be efficient and inefficient in this context, respectively. Conversely, in another context, the SUT may contain an abundance of these programming constructs, and so assertions may be inefficient, but the feature extractors available in this context may be efficient.

Table 6 A summary of the efficiency data for each technique, based on Sections 3 to 7

Technique	Examples of reasons that may explain why the efficiency of this technique might vary in different contexts.
N-Version Testing	RIs must be developed to replicate the functionality of the SUT, but they do not necessarily have to mimic other quality attributes, including efficiency. Thus, in some situations, one might develop an RI to be equally (or more) efficient to the SUT, but in another situation, one might opt to disregard efficiency completely.
Metamorphic Testing	Different MRs have different MTG sizes. Therefore, MT's efficiency in a particular context will be partly determined by the MTG sizes of the MRs that are applied in that context.
Assertions	Assertions can be inefficient at testing large arrays. Thus, the overall efficiency of Assertions in a particular context, will be determined by the number of large arrays in the SUT that must be checked by the technique.
Machine Learning	Some feature extractors are more efficient than others; since the appropriate choice of feature extractors is domain specific, the efficiency of ML may vary in different domains.
Statistical Hypothesis Testing	There is a trade-off between efficiency and result stability (which is determined by sample size). In one situation, the tester might require greater result stability than in another and thus might have to sacrifice efficiency to a greater extent in such a situation.

8.3 Usability

Table 7 demonstrates that the required effort to apply each technique can vary. Techniques may differ in terms of the contexts in which they are difficult to use. For instance, it may not be possible to obtain an RI via component harvesting for the SUT, and so manual construction of an RI may be necessary in a certain context. In the same context, all of the assumptions of a statistic being used in SHT may be satisfied by the data, so data transformation tasks are unnecessary. N-version Testing may require substantial effort in such a scenario, but SHT may not. The converse is also possible.

Table 7 also shows that the required expertise for different techniques also varies. Thus, one's choice of technique may partly depend on the expertise currently available. For example, if one lacks knowledge about machine learning, but has an adequate understanding of statistics, then one may be more inclined to select Statistical Hypothesis Testing, instead of Machine Learning oracles.

9 Related work

Although several Systematic Literature Reviews that target associated areas exist, each one explores the subject matter from a different perspective and thus offers a distinct contribution. For example, many systematic reviews had different scopes, which means they surveyed different sets of papers. For example, Kanewala and Bieman (2014), Nardi and Delamaro (2011), and Baresi and Young (2001) had a more constrained scope; they were restricted to Scientific Software, Dynamical Systems, and specification- and model-based testing, respectively. Harman et al. (2013) had a wider scope, e.g. they accounted for non-automated solutions like crowd sourcing. However, they had a different relevance criteria and search strategy, so their systematic review procured different studies.

Table 7 A summary of the usability data for each technique, based on Sections 3 to 7

Technique	Examples of reasons that may explain why the usability of this technique might vary in different contexts.	Skills and knowledge that might be required to use this technique.
N-Version Testing	In some situations, it might be necessary to implement an RI from scratch, but this may not be necessary in other situations.	The capability to write programs in different programming languages.
Metamorphic Testing	The tester may have to spend additional time and effort studying the domain to acquire domain knowledge in situations in which a domain expert is not available.	Requires domain knowledge.
Assertions	The amount of effort required to write assertions in a given context will depend on the programming language being used in that context.	Requires domain knowledge.
Machine Learning	Certain ML tasks are necessary in some situations, but are unnecessary in others, e.g. labelling dataset items.	Knowledge about machine learning.
Statistical Hypothesis Testing	Tasks like data transformation may be necessary in some situations, but not others.	Knowledge about statistics.

Different systematic reviews also conducted different types of synthesis. Harman et al. (2013), Kanewala and Bieman (2014), and Nardi and Delamaro (2011) conducted a higher level synthesis, which means their synthesis was effective for finding high level research opportunities, e.g. measurements for oracles (Harman et al. 2013), but less capable of identifying lower level research opportunities like a technique's relationship with specific fault types. Baresi and Young (2001) performed a low level synthesis, but the nature of their data is different, e.g. they explored multiple specification languages from a high level view, instead of a finer grained inspection of issues that generalise to all specifications. Finally, some systematic reviews have additional or different objectives. For example, Pezzè and Zhang (2014) and Oliveira et al. (2014a) endeavoured to establish a taxonomy to classify oracles and Harman et al. (2013) examined trends in research on the oracle problem.

Since our Mapping Study takes a unique perspective on the Oracle Problem in terms of the combination of scope, type of synthesis and objectives, it also offers a distinct contribution. In particular, our Mapping Study surveyed the literature on automated testing techniques that can detect functional software faults in non-testable systems. It also presented a series of discussions about each technique, from different perspectives like effectiveness and usability, performed a set of comparisons between these techniques, and identified research opportunities.

10 Threats to validity

This section outlines the main threats to validity and how they were mitigated. Threats are organised by Mapping Study phase.

10.1 Search

Since the first author was unfamiliar with the problem domain at the outset, relevance misclassifications were possible. To reduce this possibility, edge case papers were conservatively kept for more detailed analysis, after more knowledge had been accrued.

Many of the titles and abstracts did not give sufficient information about the true intent or scope of the paper, which may have led to misclassifications. Authors of known relevant papers were emailed with our list of their relevant papers and requested to confirm comprehensiveness. This reduced the impact of this threat.

Another threat is the restrictions placed on the search, e.g. number of research repositories. These were necessary to retain feasibility. To reduce the impact of these restrictions, we applied several other search strategies, e.g. perusing reference lists.

The search facilities offered by many repositories were flawed, which means they may not have returned all relevant studies. Where necessary, a series of workarounds were used to address this problem, e.g. using Google's "site:" function for ACM DL.

There are also threats to repeatability; web content is ever growing, and thus, the ranking of web pages are ever changing, which means that 50 consecutive irrelevant results may appear prematurely in comparison to the first search, or after significantly more results have been examined.

Including grey literature is an important step to combatting publication bias (Kitchenham 2007) and obtaining cutting edge research. We used research repositories like Google and Citeseerx to obtain such literature.

Determining the relevance of a paper is a subjective task. To reduce subjectivity, an inter-rater reliability test was conducted independently by two researchers on the Relevance Inclusion and Exclusion Criteria, on a sample of 12 papers. The results of this test were used to increase the precision of our criteria.

10.2 Data extraction

The nature of the data being captured was broad, and none of the available data extraction forms were flexible enough to capture all of the important data. Thus, a data extraction form was specifically developed for this Mapping Study, with appropriate inbuilt flexibility.

Some of the papers did not report all of the data that was necessary to complete the data extraction form, and we were unable to elicit some of this data from the authors of these papers. In such cases, it was necessary to make assumptions about the data. Although these assumptions were informed, there is a chance that they may have been incorrect.

10.3 Quality criteria

None of the available quality instruments were suitable; adoption of inappropriate quality instruments may lead to inaccurate classifications. Thus, an appropriate quality instrument was developed. The design of our quality instrument was based on the guidelines of Kitchenham (Kitchenham 2007), and took inspiration from 27 examples of quality instruments, and domain knowledge.

Measuring the quality of a paper involves some degree of subjectivity. To that end, two researchers independently conducted a test of inter-rater reliability on the quality instrument, on a sample of 12 papers. We used the results of this test to fine tune our quality criteria.

10.4 Throughout the process

Many decisions were necessarily subjective; several practices were adopted to decrease potential bias introduced through subjectivity. For example, as mentioned above, inter-rater reliability tests were conducted on several critical, subjective parts of the process. The review protocol was also defined prior to starting the Mapping Study, which enabled most subjective decisions to be taken before the data had been explored.

Additionally, where possible, subjectivity in processes was reduced through careful design, e.g. the relevance Inclusion and Exclusion Criteria are based on relatively objective guidelines.

We contacted the authors of the papers that were covered by the mapping study, at various stages of the process, by email, to elicit information and/or provide confirmation on various issues. We found that, in some cases, it was not possible to contact the author, and that a large proportion of the authors did not reply (an author is not considered to have replied if the author did not reply within a month of the last email that was sent). Given that there was such a large number of authors, human error is also possible i.e. we may have failed to email a small number of them. Additionally, even though many of the authors did reply, some of their responses only addressed a subset of the issues. This could affect our results, e.g. people that we did not establish contact with might have had a paper that could have been included in the mapping study, or had people addressed all of the issues, making certain assumptions about their work would not have been necessary.

11 Conclusion

In this paper, we conducted a mapping study on automated testing techniques that can detect functional software faults in non-testable systems. In particular, five techniques were considered—N-version Testing, Metamorphic Testing, Assertions, Machine Learning, and Statistical Hypothesis Testing.

A series of discussions revolving around issues like effectiveness, efficiency, and usability were presented about each technique, and these techniques were compared against each other based on these issues. It is our hope that this material will be a useful resource for researchers that are attempting to familiarise themselves with/navigate the field. We have embedded our own insights, that emerged from an analysis of these discussions and comparisons, throughout the material. We therefore believe that the material will also be beneficial for researchers that are already accustomed to the field. Finally, we envisage that the material could assist practitioners in selecting the most apposite technique for their context, as well as help them make an informed decision on the most beneficial method of applying the chosen technique for their particular situation. To exemplify the latter, a practitioner might have decided to use Metamorphic Testing and have tight deadlines; they could consult Section 4 for strategies on improving the efficiency of Metamorphic Testing, e.g. reducing an MR's MTG size and understanding the ramifications of using these strategies—in this case, reducing MTG size might reduce the effectiveness of Metamorphic Testing.

The aforementioned material also highlighted several potential research opportunities, which may serve to steer the direction of future research endeavours. These include the opportunity for new testing techniques that can tolerate coincidental correctness in non-testable systems, more empirical studies that explore the effectiveness of Assertions in the context of the oracle problem, and the development of context specific guidelines on how each technique should be used. It is our hope that this Mapping Study will raise awareness of these research opportunities in the research community and that researchers will peruse them. Researchers may use the results of the Mapping Study to increase confidence in the novelty of the contributions they might make by perusing these research opportunities.

Acknowledgements We would like to thank Professors Martin Shepperd and Tracy Hall for their insightful discussions on Mapping Studies. We would also like to express our gratitude to all of the authors of the papers that were processed by this Mapping Study, who assisted us through email correspondence. Finally, we would like to thank the reviewers for their helpful comments. This work was funded by the Department of Computer Science (CEDPS) at Brunel University.

References

Androutsopoulos, K., Clark, D., Dan, H., Hierons, R.M., Harman, M. (2014). An analysis of the relationship between conditional entropy and failed error propagation in software testing. In *Proceedings of the 36th international conference on software engineering* (pp. 573–583). New York: ACM.

Araujo, W., Briand, L.C., Labiche, Y. (2014). On the effectiveness of contracts as test oracles in the detection and diagnosis of functional faults in concurrent object-oriented software. *IEEE Transactions on Software Engineering*, *40*, 971–992.

Assi, R.A., Masri, W., Zaraket, F. (2016). UCov: a user-defined coverage criterion for test case intent verification. *Software Testing, Verification and Reliability*, *26*(6), 1–32.

Atkinson, C., Hummel, O., Janjic, W. (2011). Search-enhanced testing: NIER track, IEEE, Hawaii.

Baresi, L., & Young, M. (2001). Test oracles. Tech. Rep. CIS-TR-01-02, University of Oregon.

Barnett-Page, E., & Thomas, J. (2009). Methods for the synthesis of qualitative research: a critical review. *BMC Medical Research Methodology*, *9*(1), 1–26.

Brilliant, S.S., Knight, J.C., Ammann, P.E. (1990). On the performance of software testing using multiple versions. In *Proceedings of the 20th international symposium on fault-tolerant computing (FTCS-20)* (pp. 408–415). Newcastle Upon Tyne: IEEE.

Cao, Y., Zhou, Z.Q., Chen, T.Y. (2013). On the correlation between the effectiveness of metamorphic relations and dissimilarities of test case executions. In *Proceedings of the 13th international conference on quality software (QSIC)* (pp. 153–162). Najing: IEEE.

Chan, W.K., Cheung, S.C., Ho, J.CF., Tse, T.H. (2006). Reference models and automatic oracles for the testing of mesh simplification software for graphics rendering. In *Proceedings of the 30th annual international computer software and applications conference (COMPSAC)* (pp. 429–438). Chicago: IEEE.

Chan, W.K., Cheung, S.C., Leung, K.RPH. (2007). A metamorphic testing approach for online testing of service-oriented software applications. *International Journal of Web Services Research*, *4*(2), 61–81.

Chan, W.K., Cheung, S.C., Ho, J.CF., Tse, T.H. (2009). PAT: a pattern classification approach to automatic reference oracles for the testing of mesh simplification programs. *Journal of Systems and Software*, *82*(3), 422–434.

Chan, W.K., Ho, J.CF., Tse, T.H. (2010). Finding failures from passed test cases: improving the pattern classification approach to the testing of mesh simplification programs. *Software Testing, Verification and Reliability*, *20*(2), 89–120.

Chen, T.Y., Cheung, S.C., Yiu, S.M. (1998b). Metamorphic testing: a new approach for generating next test cases. Tech. Rep. HKUST-CS98-01, Hong Kong University of Science and Technology.

Chen, T.Y., Tse, T.H., Quan Zhou, Z. (2003b). Fault-based testing without the need of oracles. *Information and Software Technology*, *45*(1), 1–9.

Chen, T.Y., Huang, D.H., Tse, T.H., Zhou, Z.Q. (2004a). Case studies on the selection of useful relations in metamorphic testing. In *Proceedings of the 4th Ibero-American symposium on software engineering and knowledge engineering (JIISIC)* (pp. 569–583). Australia: Polytechnic University of Madrid.

Chen, J., Kuo, F.C., Xie, X., Wang, L. (2014c). A cost-driven approach for metamorphic testing. *Journal of Software*, *9*(9), 2267–2275.

Chen, T.Y., Ho, J.WK., Liu, H., Xie, X. (2009a). An innovative approach for testing bioinformatics programs using metamorphic testing. *BMC Bioinformatics*, *10*(1), 1–12.

Chen, T.Y., Kuo, F.C., Merkel, R., Tam, W.K. (2009c). Testing an open source suite for open queuing network modelling using metamorphic testing technique. In *Proceedings of the 14th IEEE international conference on engineering of complex computer systems* (pp. 23–29). Potsdam: IEEE.

Chen, T.Y., Tse, T.H., Zhou, Z.Q. (2011b). Semi-proving: an integrated method for program proving, testing, and debugging. *IEEE Transactions on Software Engineering*, *37*(1), 109–125.

Chen, T.Y., Poon, P.L., Xie, X. (2016). METRIC: METamorphic relation identification based on the category-choice framework. *Journal of Systems and Software*, *116*, 177–190.

Clark, D., & Hierons, R.M. (2012). Squeeziness: an information theoretic measure for avoiding fault masking. *Information Processing Letters*, *112*(8-9), 335–340.

Cruzes, D.S., & Dyba, T. (2011). Recommended steps for thematic synthesis in software engineering. In *Proceedings of the international symposium on empirical software engineering and measurement (ESEM)* (pp. 275–284). Alberta: IEEE.

Da Silva, F.QB., Cruz, S.SJO., Gouveia, T.B., Capretz, L.F. (2013). Using meta-ethnography to synthesize research: a worked example of the relations between personality and software team processes. In *Proceedings of the ACM / IEEE international symposium on empirical software engineering and measurement* (pp. 153–162). Maryland: IEEE.

Davis, M.D., & Weyuker, E.J. (1981). Pseudo-oracles for non-testable programs. In *Proceedings of the ACM 1981 Conference* (pp. 254–257). New York: ACM.

Delamaro, M.E., Nunes, F.LS., Oliveira, R.AP. (2013). Using concepts of content-based image retrieval to implement graphical testing oracles. *Software Testing, Verification and Reliability*, *23*(3), 171–198.

Dong, G., Nie, C., Xu, B., Wang, L. (2007). An effective iterative metamorphic testing algorithm based on program path analysis. In *Proceedings of the 7th international conference on quality software (QSIC)* (pp. 292–297). China: IEEE.

Dong, G., Guo, T., Zhang, P. (2013). Security assurance with program path analysis and metamorphic testing. In *Proceedings of the 4th IEEE international conference on software engineering and service science (ICSESS)* (pp. 193–197). Beijing: IEEE.

Frounchi, K., Briand, L.C., Grady, L., Labiche, Y., Subramanyan, R. (2011). Automating image segmentation verification and validation by learning test oracles. *Information and Software Technology*, *53*(12), 1337–1348.

Gotlieb, A. (2003). Exploiting symmetries to test programs. In *Proceedings of the 14th international symposium on software reliability engineering (ISSRE)* (pp. 365–374). Colorado: IEEE.

Gotlieb, A., & Botella, B. (2003). Automated metamorphic testing. In *Proceedings of the 27th annual international computer software and applications conference (COMPSAC)* (pp. 34–40). Dallas: IEEE.

Guderlei, R., & Mayer, J. (2007a). Statistical metamorphic testing testing programs with random output by means of statistical hypothesis tests and metamorphic testing. In *Proceedings of the 7th international conference on quality software (QSIC)* (pp. 404–409). Oregon: IEEE.

Guderlei, R., & Mayer, J. (2007b). Towards automatic testing of imaging software by means of random and metamorphic testing. *International Journal of Software Engineering and Knowledge Engineering*, *17*(6), 757–781.

Guderlei, R., Mayer, J., Schneckenburger, C., Fleischer, F. (2007). Testing randomized software by means of statistical hypothesis tests. In *Proceedings of the 4th international workshop on software quality assurance: in conjunction with the 6th ESEC/FSE joint meeting* (pp. 46–54). New York: ACM.

Harman, M., McMinn, P., Shahbaz, M., Yoo, S. (2013). A comprehensive survey of trends in oracles for software testing. Tech. Rep. TR-09-03, University of Sheffield.

Higgins, J., & Green, S. (2011). Cochrane handbook for systematic reviews of interventions. The Cochrane Collaboration.

Hoffman, D. (1999). Heuristic test oracles. *Software Testing & Quality Engineering* 29–32.

Hummel, O., Atkinson, C., Brenner, D., Keklik, S. (2006). Improving testing efficiency through component harvesting. Tech. rep., University of Mannheim.

Janjic, W., Barth, F., Hummel, O., Atkinson, C. (2011). Discrepancy discovery in search-enhanced testing. In *Proceedings of the 3rd international workshop on search-driven development: users, infrastructure, tools, and evaluation* (pp. 21–24). New York: ACM.

Jiang, M., Chen, T.Y., Kuo, F.C., Zhou, Z.Q., Ding, Z. (2014). Testing model transformation programs using metamorphic testing. In *Proceedings of the 26th international conference on software engineering and knowledge engineering (SEKE)* (pp. 94–99). Vancouver: Knowledge Systems Institute Graduate School.

Kanewala, U., & Bieman, J.M. (2013a). Techniques for testing scientific programs without an oracle. In *Proceedings of the 5th international workshop on software engineering for computational science and engineering (SE-CSE)* (pp. 48–57). California: IEEE.

Kanewala, U., & Bieman, J.M. (2013b). Using machine learning techniques to detect metamorphic relations for programs without test oracles. In *Proceedings of the 24th international symposium on software reliability engineering (ISSRE)* (pp. 1–10). USA: IEEE.

Kanewala, U., & Bieman, J.M. (2014). Testing scientific software: a systematic literature review. *Information and Software Technology*, *56*(10), 1219–1232.

Kanewala, U., Bieman, J.M., Ben-Hur, A. (2014). Predicting metamorphic relations for testing scientific software: a machine learning approach using graph kernels. *Journal of Software Testing, Verification and Reliability* 1–25.

Kitchenham, B. (2007). Guidelines for performing systematic literature reviews in software engineering. Tech. Rep. EBSE-2007-01, Keele University.

Kuo, F.C., Zhou, Z.Q., Ma, J., Zhang, G. (2010). Metamorphic testing of decision support systems: a case study. *IET Software*, *4*(4), 294–301.

Kuo, F.C., Liu, S., Chen, T.Y. (2011). Testing a binary space partitioning algorithm with metamorphic testing. In *Proceedings of the 2011 ACM symposium on applied computing* (pp. 1482–1489). New York: ACM.

Liu, H., Liu, X., Chen, T.Y. (2012). A new method for constructing metamorphic relations. In *Proceedings of the 12th international conference on quality software (QSIC)* (pp. 59–68). Shaanxi: IEEE.

Liu, H., Kuo, F.C., Towey, D., Chen, T.Y. (2014a). How effectively does metamorphic testing alleviate the oracle problem? *IEEE Transactions on Software Engineering*, *40*(1), 4–22.

Liu, H., Yusuf, I.I., Schmidt, H.W., Chen, T.Y. (2014b). Metamorphic fault tolerance: an automated and systematic methodology for fault tolerance in the absence of test oracle. In *Proceedings of the 36th international conference on software engineering* (pp. 420–423). New York: ACM.

Lozano, R.C. (2010). *Constraint programming for random testing of a trading system*. Stockholm: PhD thesis Polytechnic University of Valencia.

Manolache, L.I., & Kourie, D.G. (2001). Software testing using model programs. *Software: Practice and Experience*, *31*(13), 1211–1236.

Mayer, J. (2005). On testing image processing applications with statistical methods. In *Proceedings of software engineering, lecture notes in informatics* (pp. 69–78). Bonn.

Mayer, J., & Guderlei, R. (2004). Test oracles using statistical methods. In *Proceedings of the 1st international workshop on software quality* (pp. 179–189). Springer.

Mayer, J., & Guderlei, R. (2006). An empirical study on the selection of good metamorphic relations. In *Proceedings of the 30th annual international computer software and applications conference (COMPSAC)* (pp. 475–484). Chicago: IEEE.

McMinn, P. (2009). Search-based failure discovery using testability transformations to generate pseudo-oracles. In *Proceedings of the 11th annual conference on genetic and evolutionary computation* (pp. 1689–1696). New York: ACM.

Merkel, R., Wang, D., Lin, H., Chen, T.Y. (2011). Automatic verification of optimization algorithms: a case study of a quadratic assignment problem solver. *International Journal of Software Engineering and Knowledge Engineering*, *21*(2), 289–307.

Mishra, K.S., Kaiser, G.E., Sheth, S.K. (2013). Effectiveness of teaching metamorphic testing, part II. Tech. Rep. CUCS-022-13, Columbia University.

Monisha, T.R., & Chamundeswari, A. (2013). Automatic verification of test oracles in functional testing. In *Proceedings of the 4th international conference on computing, communications and networking technologies (ICCCNT)* (pp. 1–4). Tiruchengode: IEEE.

Murphy, C., & Kaiser, G.E. (2008). Improving the dependability of machine learning applications. Tech. Rep. cucs-049-08, Columbia University, 116th and Broadway, New York, NY 10027.

Murphy, C., & Kaiser, G.E. (2009). Metamorphic runtime checking of non-testable programs. Tech. Rep. cucs-042-09, Columbia University, 116th and Broadway, New York, NY 10027.

Murphy, C., & Kaiser, G.E. (2010). Automatic detection of defects in applications without test oracles. Tech. Rep. CUCS-027-10, Columbia University, 116th and Broadway, New York, NY 10027.

Murphy, C., Kaiser, G.E., Hu, L. (2008). Properties of machine learning applications for use in metamorphic testing. In *Proceedings of the 20th international conference on software engineering and knowledge engineering (SEKE)* (pp. 867–872). California: SEKE.

Murphy, C., Shen, K., Kaiser, G. (2009a). Automatic system testing of programs without test oracles. In *Proceedings of the 18th international symposium on software testing and analysis* (pp. 189–200). New York: ACM.

Murphy, C., Shen, K., Kaiser, G. (2009b). Using JML runtime assertion checking to automate metamorphic testing in applications without test oracles. In *Proceedings of the international conference on software testing verification and validation (ICST)* (pp. 436–445). Colorado: IEEE.

Murphy, C., Raunak, M.S., King, A., Chen, S., Imbriano, C., Kaiser, G., Lee, I., Sokolsky, O., Clarke, L., Osterweil, L. (2011). On effective testing of health care simulation software. In *Proceedings of the 3rd workshop on software engineering in health care* (pp. 40–47). New York: ACM.

Murphy, C., Kaiser, G.E., Bell, J.S., Su, F.h. (2013). Metamorphic runtime checking of applications without test oracles. Tech. Rep. CUCS-023-13, Columbia University, 116th and Broadway, New York, NY 10027.

Nardi, P.A., & Delamaro, M.E. (2011). Test oracles associated with dynamical systems models. Tech. Rep. RT 362, Universidade De Sao Paulo.

Núñez, A., & Hierons, R.M. (2015). A methodology for validating cloud models using metamorphic testing. *Annals of Telecommunications - Annales des Télécommunications*, *70*(3), 127–135.

Oliveira, R.AP., Kanewala, U., Nardi, P.A. (2014a). Automated test oracles: state of the art, taxonomies, and trends. In Memon, A. (Ed.) *Advances in computers* (pp. 113–199): Elsevier.

Oliveira, R.AP., Memon, A.M., Gil, V.N., Nunes, F.L., Delamaro, M. (2014b). An extensible framework to implement test oracles for non-testable programs. In *Proceedings of the 26th international conference on software engineering and knowledge engineering (SEKE)* (pp. 199–204). Knowledge Systems Institute: Vancouver.

Paleari, R., Martignoni, L., Roglia, G.F., Bruschi, D. (2010). N-version disassembly: differential testing of x86 disassemblers. In *Proceedings of the 19th international symposium on software testing and analysis* (pp. 265–274). New York: ACM.

Patel, K., & Hierons, R.M. (2015). The interlocutory tool box: techniques for curtailing coincidental correctness. Tech. rep., Brunel University.

Pezzè, M., & Zhang, C. (2014). Automated test oracles: a survey. In Memon, A. (Ed.) *Advances in computers* (pp. 1–48): Elsevier.

Poon, P.L., Kuo, F.C., Liu, H., Chen, T.Y. (2014). How can non-technical end users effectively test their spreadsheets? *Information Technology and People*, *27*(4), 440–462.

Popay, J., Roberts, H., Sowden, A., Petticrew, M., Arai, L., Rodgers, M., Britten N (2006). *Guidance on the conduct of narrative synthesis in systematic reviews: a product from the ESRC methods program.* Lancaster University.

Segura, S., Hierons, R.M., Benavides, D., Ruiz-Cortés, A. (2011). Automated metamorphic testing on the analyses of feature models. *Information and Software Technology*, *53*(3), 245–258.

Ševčíková, H., Borning, A., Socha, D., Bleek, W.G. (2006). Automated testing of stochastic systems: a statistically grounded approach. In *Proceedings of the 2006 international symposium on software testing and analysis* (pp. 215–224). New York: ACM.

Shepperd, M. (2013). Combining evidence and meta-analysis in software engineering. In Lucia AD, & Ferrucci, F. (Eds.) *Software engineering* (pp. 46–70). Berlin: Springer.

Sim, K.Y., Wong, D.ML., Hii, T.Y. (2013). Evaluating the effectiveness of metamorphic testing on edge detection programs. *International Journal of Innovation Management and Technology*, *4*(1), 6–10.

Sim, K.Y., Low, C.S., Kuo, F.C. (2014). Eliminating human visual judgment from testing of financial charting software. *Journal of Software*, *9*(2), 298–312.

Tiwari, S., Mishra, K.K., Kumar, A., Misra, A.K. (2011). Spectrum-based fault localization in regression testing. In *Proceedings of the 8th international conference on information technology: new generations (ITNG)* (pp. 191–195). Las Vegas: IEEE.

Weyuker, E.J. (1982). On testing non-testable programs. *The Computer Journal*, *25*(4), 465–470.

Wu, P. (2005). In *Proceedings of the 29th annual international computer software and applications conference (COMPSAC)* (pp. 19–24). Edinburgh: IEEE.

Yoo, S. (2010). Metamorphic testing of stochastic optimisation. In *Proceedings of the 3rd international conference on software testing, verification, and validation workshops (ICSTW)* (pp. 192–201). Paris: IEEE.

Zhang, Z., Chan, W.K., Tse, T.H., Hu, P. (2009). Experimental study to compare the use of metamorphic testing and assertion checking. *Journal of Software*, *20*(10), 2637–2654.

Further Reading

Arcaini, P., Gargantini, A., Riccobene, E. (2013). Combining model-based testing and runtime monitoring for program testing in the presence of nondeterminism. In *Proceedings of the 6th international conference on software testing, verification and validation workshops (ICSTW)* (pp. 178–187). Luxembourg: IEEE.

Asrafi, M., Liu, H., Kuo, F.C. (2011). On testing effectiveness of metamorphic relations: a case study. In *Proceedings of the 5th international conference on secure software integration and reliability improvement (SSIRI)* (pp. 147–156). Jeju Island: IEEE.

Barr, E.T., Harman, M., McMinn, P., Shahbaz, M., Yoo, S. (2015). The oracle problem in software testing: a survey. *IEEE Transactions on Software Engineering*, *41*(5), 507–525.

Barus, A.C., Chen, T.Y., Grant, D., Kuo, F.C., Lau, M.F. (2011). Testing of heuristic methods: a case study of greedy algorithm. In Huzar Z, Koci, R., Meyer, B., Walter, B., Zendulka, J. (Eds.) *Software Engineering Techniques* (pp. 246–260). Berlin: Springer.

Batra, G., & Sengupta, J. (2011). An efficient metamorphic testing technique using genetic algorithm. In Dua, S., Sahni, S., Goyal, D.P. (Eds.) *Information intelligence, systems, technology and management* (pp. 180–188). Berlin: Springer.

Chan, W.K., & Tse, T.H. (2013). Oracles are hardly attain'd, and hardly understood: confessions of software testing researchers. In *Proceedings of the 13th international conference on quality software (QSIC)* (pp. 245–252). Najing: IEEE.

Chan, F.T., Chen, T.Y., Cheung, S.C., Lau, M.F., Yiu, S.M. (1998). Application of metamorphic testing in numerical analysis. In *Proceedings of the IASTED international conference on software engineering* (pp. 191–197). Las Vegas: ACTA Press.

Chen, H.Y. (1999). The application of an algebraic design method to deal with oracle problem in object-oriented class level testing. In *Proceedings of the international conference on systems, man, and cybernetics (SMC)* (pp. 928–932). Tokyo: IEEE.

Chen, T.Y. (2010). Metamorphic testing: a simple approach to alleviate the oracle problem. In *Proceedings of the 5th IEEE international symposium on service oriented system engineering (SOSE)* (pp. 1–2). Nanjing: IEEE.

Chen, H.Y., Tse, T.H., Chan, F.T., Chen, T.Y. (1998a). In black and white: an integrated approach to class-level testing of object-oriented programs. *ACM Transactions on Software Engineering and Methodology (TOSEM,)* *7*(3), 250–295.

Chen, H.Y., Tse, T.H., Deng, Y.T. (2000). ROCS: an object-oriented class-level testing system based on the relevant observable contexts technique. *Information and Software Technology*, *42*(10), 677–686.

Chen, T.Y., Feng, J., Tse, T.H. (2002). Metamorphic testing of programs on partial differential equations: a case study. In *Proceedings of the 26th annual international computer software and applications conference (COMPSAC)* (pp. 327–333). Oxford: IEEE.

Chen, T.Y., Kuo, F.C., Tse, T.H., Zhou, Z.Q. (2003a). Metamorphic testing and beyond. In *Proceedings of the 11th annual international workshop on software technology and engineering practice* (pp. 94–100). Amsterdam: IEEE.

Chen, T.Y., Kuo, F.C., Liu, Y., Tang, A. (2004b). Metamorphic testing and testing with special values. In *Proceedings of the 5th ACIS international conference on software engineering, artificial intelligence, networking and parallel/distributed computing* (pp. 128–134). Beijing: ACIS.

Chen, T.Y., Kuo, F.C., Liu, H., Wang, S. (2009b). Conformance testing of network simulators based on metamorphic testing technique. In *Proceedings of the joint 11th IFIP WG 6.1 international conference FMOODS and 29th IFIP WG 6.1 international conference FORTE* (pp. 243–248). Portugal: Springer.

Chen, T.Y., Kuo, F.C., Tam, W.K., Merkel, R. (2011a). Testing a software-based PID controller using metamorphic testing, SciTePress - Science and Technology Publications, Algarve.

Chen, L., Cai, L., Liu, J., Liu, Z., Wei, S., Liu, P. (2012a). An optimized method for generating cases of metamorphic testing. In *Proceedings of the 6th international conference on new trends in information science and service science and data mining (ISSDM)* (pp. 439–443). Taipei: IEEE.

Chen, T.Y., Kuo, F.C., Towey, D., Zhou, Z.Q. (2012b). Metamorphic testing: applications and integration with other methods: tutorial synopsis. In *Proceedings of the 12th international conference on quality software (QSIC)* (pp. 285–288). Shaanxi: IEEE.

Ding, J., & Xu, D. (2012). Model-based metamorphic testing: a case study. In *Proceedings of the 24th international conference on software engineering and knowledge engineering (SEKE)* (pp. 363–368). San Francisco Bay: Knowledge Systems Institute Graduate School.

Ding, J., Wu, T., Lu, J., Hu, X.H. (2010). Self-checked metamorphic testing of an image processing program. In *Proceedings of the 4th international conference on secure software integration and reliability improvement (SSIRI)* (pp. 190–197). Singapore: IEEE.

Ding, J., Wu, T., Wu, D., Lu, J.Q., Hu, X.H. (2011). Metamorphic testing of a monte carlo modeling program. In *Proceedings of the 6th international workshop on automation of software test* (pp. 1–7). New York: ACM.

Dong, G., Xu, B., Chen, L., Nie, C., Wang, L. (2008). Case studies on testing with compositional metamorphic relations. *Journal of Southeast University*, *24*(4), 437–443.

Gore, R., Reynolds, P.FJ., Kamensky, D. (2011). Statistical debugging with elastic predicates. In *Proceedings of the 26th IEEE/ACM international conference on automated software engineering (ASE)* (pp. 492–495). Kansas: IEEE.

Huang, S., Ji, M.Y., Hui, Z.W., Duanmu, Y.T. (2011). Detecting integer bugs without oracle based on metamorphic testing technique. *Applied Mechanics and Materials*, *121–126*, 1961–1965.

Hui, Z., Huang, S., Ren, Z., Yao, Y. (2013). Metamorphic testing integer overflow faults of mission critical program: a case study. *Mathematical Problems in Engineering* 1–6.

Jiang, M., Chen, T.Y., Kuo, F.C., Ding, Z. (2013). Testing central processing unit scheduling algorithms using metamorphic testing. In *Proceedings of the 4th IEEE international conference on software engineering and service science (ICSESS)* (pp. 530–536). Beijing: IEEE.

Just, R., & Schweiggert, F. (2011). Automating unit and integration testing with partial oracles. *Software Quality Journal*, *19*(4), 753–769.

Kamensky, D., Gore, R., Reynolds, P.FJ. (2011). Applying enhanced fault localization technology to Monte Carlo simulations. In *Proceedings of the 2011 Winter Simulation Conference (WSC)* (pp. 2798–2809). Arizona: IEEE.

Khosla, T., & Garg, S. (2011). Metamorphic testing effectiveness on trignometry. *International Journal of Computer Science and Technology*, *2*(3), 576–578.

Kim-Park, D.S., Riva, C., Tuya, J. (2009). A partial test oracle for XML query testing. In *Proceedings of the testing: academic and industrial conference on practice and research techniques (TAIC PART)* (pp. 13–20). Windsor: IEEE.

Kim-Park, D.S., Riva, C., Tuya, J. (2010). An automated test oracle for XML processing programs. In *Proceedings of the 1st international workshop on software test output validation* (pp. 5–12). New York: ACM.

Lei, Y., Mao, X., Chen, T.Y. (2013). Backward-slice-based statistical fault localization without test oracles. In *Proceedings of the 13th international conference on quality software (QSIC)* (pp. 212–221). Najing: IEEE.

Liu, H., Wang, D., Lin, H., Chen, T.Y. (2009). On the integration of metamorphic testing and model checking. In *Proceedings of the IADIS international conference applied computing* (pp. 299–302). Rome: IADIS Press.

Lu, X.L., Dong, Y.w., Luo, C. (2010). Testing of component-based software: a metamorphic testing methodology. In *Proceedings of the 7th international conference on ubiquitous intelligence computing and 7th international conference on autonomic trusted computing (UIC/ATC)* (pp. 272–276). Xian: IEEE.

Martignoni, L., Paleari, R., Roglia, G.F., Bruschi, D. (2010). Testing system virtual machines. In *Proceedings of the 19th international symposium on software testing and analysis* (pp. 171–182). New York: ACM.

Nardo, D.D., Alshahwan, N., Briand, L.C., Fourneret, E., Nakic-Alfirevic, T., Masquelier, V. (2013). Model based test validation and oracles for data acquisition systems. In *Proceedings of the 28th IEEE/ACM international conference on automated software engineering (ASE)* (pp. 540–550). California: IEEE.

Oliveira, R.AP., Delamaro, M.E., Nunes, F.LS. (2011). Exploring the OFIm framework to support the test of programs with GUIs. Tech. rep., University of Sao Paulo.

Padgham, L., Zhang, Z., Thangarajah, J., Miller, T. (2013). Model-based test oracle generation for automated unit testing of agent systems. *IEEE Transactions on Software Engineering*, *39*(9), 1230–1244.

Poutakidis, D., Padgham, L., Winikoff, M. (2002). Debugging multi-agent systems using design artifacts: the case of interaction protocols. In *Proceedings of the 1st international joint conference on autonomous agents and multiagent systems: part 2* (pp. 960–967). New York: ACM.

Poutakidis, D., Winikoff, M., Padgham, L., Zhang, Z. (2009). Debugging and testing of multi-agent systems using design artefacts. In *Multi-agent programming* (pp. 215–258). Boston: Springer.

Pullum, L.L., & Ozmen, O. (2012). Early results from metamorphic testing of epidemiological models. In *Proceedings of the ASE/IEEE international conference on biomedical computing (BioMedCom)* (pp. 62–67). Washington: IEEE.

Rao, P., Zheng, Z., Chen, T.Y., Wang, N., Cai, K. (2013). Impacts of test suite's class imbalance on spectrum-based fault localization techniques. In *Proceedings of the 13th international conference on quality software* (pp. 260–267). Najing: IEEE.

Roper, M. (2009). Artifical immune systems, danger theory, and the oracle problem. In *Proceedings of the testing: academic and industrial conference on practice and research techniques (TAIC PART)* (pp. 125–126). Windsor: IEEE.

Sadi, M.S., Kuo, F.C., Ho, J.WK., Charleston, M.A., Chen, T.Y. (2011). Verification of phylogenetic inference programs using metamorphic testing. *Journal of Bioinformatics and Computational Biology*, *9*(6), 729–747.

Satpathy, M., Butler, M., Leuschel, M., Ramesh, S. (2007). Automatic testing from formal specifications. In *Tests and proofs* (pp. 95–113): Springer.

Segura, S., Durán, A., Sánchez, A.B., Le Berre, D., Lonca, E., Ruiz-Cortés, A. (2013). Automated metamorphic testing on the analysis of software variability. Tech. Rep. SA-2013-TR-03, University of Seville.

Sim, K.Y., Pao, W.KS., Lin, C. (2005). Metamorphic testing using geometric interrogation technique and its application. In Proceedings of the 2nd *international conference of electrical engineering/electronics, computer, telecommunications, and information technology* (ECTI) (pp. 91–95).

Sim, K.Y., Low, C.S., Kuo, F.C. (2010). Detecting faults in technical indicator computations for financial market analysis. In *Proceedings of the 2nd international conference on information science and engineering (ICISE)* (pp. 2749–2754). Hangzhou: IEEE.

Singh, G., & Singh, G. (2012). An automated metamorphic testing technique for designing effective metamorphic relations. In Parashar M, Kaushik, D., Rana, O.F., Samtaney, R., Yang, Y., Zomaya, A. (Eds.) *Contemporary computing* (pp. 152–163). Berlin: Springer.

Sánchez, A.B., & Segura, S. (2012). Automated testing on the analysis of variability-intensive artifacts: an exploratory study with SAT solvers. Tech. rep., University of Seville.

Tao, Q., Wu, W., Zhao, C., Shen, W. (2010). An automatic testing approach for compiler based on metamorphic testing technique. In *Proceedings of the 17th Asia Pacific software engineering conference (APSEC)* (pp. 270–279). Sydney: IEEE.

Tse, T.H. (2005). Research directions on model-based metamorphic testing and verification. In *Proceedings of the 29th annual international computer software and applications conference (COMPSAC)* (p. 1). Edinburgh: IEEE.

Tse, T.H., Chen, T.Y., Zhou, Z. (2000). Testing of large number multiplication functions in cryptographic systems. In *Proceedings of the 1st Asia-Pacific conference on quality software* (pp. 89–98). Hong Kong: IEEE.

Tse, T.H., Lau, F.CM., Chan, W.K., Liu, P.CK., Luk, C.KF. (2007). Testing object-oriented industrial software without precise oracles or results. *Communications of the ACM*, *50*(8), 78–85.

Weyuker, E.J. (1980). The oracle assumption of program testing. In *Proceedings of the 13th international conference on system sciences (ICSS)* (pp. 44–49). Hawaii: Western Periodicals.

Wu, P., Shi, X.c., Tang, J.j., Lin, H.m., Chen, T.Y. (2005). Metamorphic testing and special case testing: a case study. *Journal of Software*, *16*(7), 1210–1220.

Xie, X., Ho, J.WK., Murphy, C., Kaiser, G., Xu, B., Chen, T.Y. (2011). Testing and validating machine learning classifiers by metamorphic testing. *Journal of Systems and Software*, *84*(4), 544–558.

Xie, X., Wong, W.E., Chen, T.Y., Xu, B. (2013). Metamorphic slice: an application in spectrum-based fault localization. *Information and Software Technology*, *55*(5), 866–879.

Yao, Y., Huang, S., Ji, M. (2012). Research on metamorphic testing for oracle problem of integer bugs. In Jin D, & Lin, S. (Eds.) *Advances in computer science and information engineering* (pp. 95–100). Berlin: Springer.

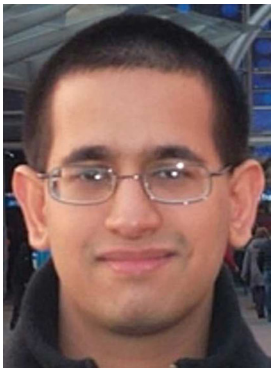
Yi, Y., Changyou, Z., Song, H., Zhengping, R. (2013). Research on metamorphic testing: a case study in integer bugs detection. In *Proceedings of the 4th international conference on intelligent systems design and engineering applications* (pp. 488–493). China: IEEE.

Zhang, J., Hu, X., Zhang, B. (2011). An evaluation approach for the program of association rules algorithm based on metamorphic relations. *Journal of Electronics (China)*, *28*(4-6), 623–631.

Zheng, W., Ma, H., Lyu, M.R., Xie, T., King, I. (2011). Mining test oracles of web search engines. In *Proceedings of the 26th IEEE/ACM international conference on automated software engineering (ASE)* (pp. 408–411). Kansas: IEEE.

Zhou, Z.Q., Huang, D.H., Tse, T.H., Yang, Z., Huang, H., Chen, T.Y. (2004). Metamorphic testing and its applications. In *Proceedings of the 8th international symposium on future software technology (ISFST)* (pp. 1–6). The Software Engineers Association: Xian.

Zhou, Z.Q., Tse, T.H., Kuo, F.C., Chen, T.Y. (2007). Automated functional testing of web search engines in the absence of an oracle. Tech. Rep. TR-2007-06, The University of Hong Kong.



Krishna Patel received a BSc in Computer Science at Brunel University London. He is currently a Ph.D. student and graduate teaching assistant at Brunel University London and a member of the Brunel Software Engineering Lab (BSEL).

Robert M. Hierons received a BA in Mathematics (Trinity College, Cambridge), and a Ph.D. in Computer Science (Brunel University). He then joined the Department of Mathematical and Computing Sciences at Goldsmiths College, University of London, before returning to Brunel University in 2000. He was promoted to full Professor in 2003.