Automated Test Oracles: State of the Art, Taxonomies and Trends

1Dept. of Computer Systems, University of São Paulo (ICMC/USP), São Carlos, SP, Brazil
2Computer Science Department, Colorado State University (CSU), Fort Collins, CO, USA

Rafael A.P. Oliveira
Upulee Kanewala
Paulo A. Nardi

USP/CSU – 2014
October/2014

Abstract

Test oracle methods have changed significantly over time, which has resulted in clear shifts in the research literature. Over the years, the testing techniques, strategies, and criteria utilized by researchers went through technical developments due to the improvement of technologies and programming languages. Software testing designers, known as testers, currently have several resources to increase their confidence in the software under test correctness. All of these Software Testing resources are supposed to include a mechanism to decide whether a particular execution is considered a failure or not. In Software Testing environments, this decision is the responsibility of the test oracle. Despite the evolution and adaptation of testing techniques over more than 30 years, test oracles remain a particular and relevant issue. In this chapter, using literary evidence from a pool of about 300 studies directly related to test oracles, we present a classification of test oracles based on a taxonomy that considers their source of information and notations. Based on this classification, we perform a quantitative analysis to highlight the shifts in (evolution of) research on test oracles. Exploring geographical and quantitative information, we analyzed the maturity of this field using co-authorship networks among studies published between 1978 and 2013. Further, we determine the most prolific authors and their countries, main conferences and journals, supporting tools, academic efforts, and use a comparative analysis between academia and industry. Finally, from these analyzes we draw an analytic reflection about contemporary test oracle approaches and a criticism about oracle trends.

Keywords– test oracles; oracle problem; software testing; automated test oracle; survey; scoping study; state of the art; system quality;
Contents

1 Introduction 3

2 Background 5
   2.1 Software Testing Concepts .................................................. 5
   2.2 Automated Software Testing ................................................ 7
   2.3 Test Oracles ................................................................. 9
      2.3.1 The Oracle Problem .................................................. 13
      2.3.2 Trade-off on Test Oracles ........................................... 15

3 Oracles Taxonomies 16
   3.1 Generic Classifications .................................................... 16
      3.1.1 Pseudo-oracles and Partial Oracles (Davis e Weyuker [55] and Weyuker [235]) 16
      3.1.2 Passive and Active Oracles ........................................... 17
   3.2 Specific Taxonomies ....................................................... 18
      3.2.1 Source of Oracles (Beizer [24]) .................................... 18
      3.2.2 Classification by Automation Process (Hoffman [86, 87, 88]) .... 19
      3.2.3 Oracle Categories According to Harman et al. [83] ................ 19
   3.3 A Taxonomy by Oracle Information Characteristics .................... 21
      3.3.1 Specification-based Oracles (Baresi e Young [22]) ............... 21
      3.3.2 Metamorphic Relation-based Oracles ................................ 28
      3.3.3 Machine Learning Based Oracles ................................... 29
      3.3.4 Version-based Oracles ................................................ 30

4 A Quantitative Analysis and a Mapping of Studies 31
   4.1 A Literature Review on Test Oracles .................................... 31
      4.1.1 Study Selection ....................................................... 32
      4.1.2 Study Classification ................................................ 32
   4.2 A Quantitative Analysis on Studies ..................................... 33
      4.2.1 Study Analysis per Area ............................................. 33
      4.2.2 SUT Analysis .......................................................... 36
      4.2.3 Projects Analysis .................................................... 38
      4.2.4 Demographical Analysis ............................................. 39
      4.2.5 Publication Strategies .............................................. 40
      4.2.6 Prolific Researchers ................................................ 42
   4.3 Authors’ Collaboration ..................................................... 44
   4.4 Surveys and Position Studies ............................................. 47
# Automated Test Oracles: State of the Art, Taxonomies, and Trends

by Rafael A.P. Oliveira, Upulee Kanewala and Paulo A. Nardi

(to appear) in Advances in Computers (Chapter Three), volume 95, pages 113-199, Elsevier, 2015 (published online in 2014)

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.5 Supporting Tools</td>
<td>49</td>
</tr>
<tr>
<td>4.6 Academic Efforts on Test Oracles (Ph.D. and Masters)</td>
<td>50</td>
</tr>
<tr>
<td>5 Discussions</td>
<td>51</td>
</tr>
<tr>
<td>5.1 High Level of Tools and Specification Languages</td>
<td>52</td>
</tr>
<tr>
<td>5.2 Complexity of SUT's Outputs</td>
<td>53</td>
</tr>
<tr>
<td>5.3 Generalization of Test Oracle Strategies and Properties</td>
<td>55</td>
</tr>
<tr>
<td>5.4 Trends on Test Oracle</td>
<td>56</td>
</tr>
<tr>
<td>6 Final and Concluding Remarks</td>
<td>58</td>
</tr>
</tbody>
</table>
1 Introduction

A fundamental element in the automated testing processes is the “test oracle”, which represents a method (program, function, set of data or table of values, etc.) to verify whether the SUT (Software Under Test) behaves correctly on a particular execution \[22, 92, 197\]. This element plays a decisive role in software testing processes establishing a decision about the correctness of an SUT result \[55, 56, 235\]. According to Ivory e Hearst \[100\], test oracles must support testers’ decision about the SUT behavior against an input, establishing which results/behaviors are acceptable. In this sense, test oracles are reliable sources of information that testers can trust to decide the correct test results \[28, 137, 220\].

In technical terms, a test oracle can be carried out in different ways: functions, assertions, processes, data, parallel programs, and others \[83\]. On the other hand, even a human being can play the role of test oracle deciding about the correctness of test results. Regardless of the different ways of implementation, oracles bear a fundamental responsibility for the whole testing process: they judge and report to testers whether the result of an execution is correct \[22, 86, 87\].

Developing an adequate test oracle for particular test domains offers a number of challenges. First, oracles are generally separated from the SUT and, then, one needs complex pre/post conditions to automate them in testing environments \[102\]. Second, developing test oracles involves controllability and observability issues \[10\]. This means that often it is necessary to determine external mechanisms to control and observe the functionality of the SUT, controlling and observing different inputs’ effects on SUT’s outputs. In addition, there are cases in which the tester must expend a precious amount of project time verifying test outputs in order to define whether they are acceptable. Finally, depending on the complexity of the output, in some occasions it is not possible to establish a test oracle. In these cases, the tester could explore heuristics (statistical functions or models) to provide expected results, establishing an oracle assumption.

Although much of the Software Engineering (SE) literature provides evidence about the importance of automated testing activities, in practice test oracle automation is well known as being a difficult task. During the development of an automated test environment (including a test oracle) in practical software projects, the tester deals with at least four different delicate issues. First is defining what should be tested. Second is defining what are acceptable outputs and under which conditions. Third is establishing acceptable outputs. Fourth is identifying reliable sources of information and models which one can trust as a reference for test oracles.

Identifying the correct aspects related to each issue presented previously is essential for testers. Once testers have an adequate and automated test scenario, including a mechanism to support their final decisions (oracle), the software product can be evaluated.
in a productive and effective way. Then, testers can load and run test cases automatically, without the necessity of human intervention after setting test cases, allowing automatic setting of system conditions and persistence of these conditions. This automated test scenario must include resources for comparisons between current and expected results. In other words, methods to analyze each individual test case result (pass/fail).

The current scenario in the sense of software quality reveals that there is no more space for poorly tested software systems. Despite significant improvements of software development resources, testing activities have become increasingly important and crucial to the daily life of common people. Software systems developed within the last five years reach more people in a shorter time than decades ago, so delivering non-tested software might decree the unacceptability of a product [10]. Demanding customers, ranking in software centers, customer’s reviews, etc. drive the current software systems to be literally viewed as a product and not a solution to be chosen by consumers. These aspects increase the software industry competitiveness. Therefore, developers regard the test as one of the most important activities during the software development process.

The central theme of this chapter has to do with the test oracles, which represent indispensable mechanisms in automated test environments. A test oracle is a relevant topic in SE and researchers and practitioners have studied them for more than 30 years. Among the main goals of this chapter, we present:

- A broad taxonomy about test oracles, with practical examples and detailed descriptions;
- A temporal evolution of research on test oracle;
- Demographic and quantitative information based on a pool of studies directly related to test oracles; and
- A critical view about trends and relevant issues to be faced by the researchers in the next years.

Following this introductory section, the rest of this chapter is organized as follows:

- Section 2 introduces basic concepts of software testing and presents comparative information about manual testing and automated testing. In addition, this section describes oracle definitions using a high level of information and giving practical examples;
- Section 3 describes various test oracle approaches to establish an oracle taxonomy. This information was collected and classified from scientific studies related to test oracles. Practical examples and code excerpts are presented to illustrate the oracle rule in real software testing scenarios;
• Section 4 reports the test oracle’s state-of-the-art. This section uses a pool of about 300 test oracle scientific studies to compose a survey. We explore quantitative and geographical analysis to describe the general scenario of oracle research over more than 30 years. In addition, we present similar surveys available, supporting tools, and academic dissertations available;

• Section 5 drives forward a set of oracle relevant issues that must be faced by researchers and practitioners on software testing. Using Section 4 as background, these thoughts are based on our own point of view about present and future research on test oracles; and

• Section 6 observes the final and concluding remarks addressed to test oracles in this chapter. Furthermore, this section show the relevance and importance of test oracles for automated software testing environments.

2 Background

Despite the advanced resources of software development available, programmers can make “mistakes” during the process of writing code. Such mistakes set “faults” for performing a process, method, or activity. Manifestations of faults can lead software systems to inconsistent states and then several “errors” might be revealed [169]. In essence, the execution of a defective source code leads the program to an unexpected state and that is visible by means of the system output, revealing failures [97, 142]. The central point of this chapter refers to the means for defining whether the current results agree with the expected outcomes in software testing scenarios – test oracle. This section presents a wide view of software testing concepts and, after that, it introduces the concept of test oracles in this scenario.

2.1 Software Testing Concepts

Myers et al. [169] emphasized that software testing activities can be the process, or series of processes, whose goal is running a system in order to find errors. According to Bertolino [25, 26], software testing is the execution process of a software product aiming to check whether it reaches its specifications considering the environment in which it was designed. Hunter e Strooper [95] agree that software testing is the main resource to check specifications against current behaviors. The standard way of thinking about software testing is that there are different useful definitions to describe it. In this study, we define software testing as a way to verify whether the SUT behaves in accordance with its specification through a controlled execution. Furthermore, Bertolino
[25] considers the following software testing definition as a reference framework, allowing different associations with testing concepts: “... Software testing consists of the dynamic verification of the behavior of a program on a finite set of test cases, suitably selected from the usually infinite execution domain, against the specified expected behavior ...”

In this introductory scenario, one can realize that over more than 30 years of research and practice, software testing environments have suffered significant changes and advances [10]. New software testing activities incorporate innovations in order to follow trends such as Object-Oriented (OO) [38], Aspect-Oriented (AO) [33, 70, 120], web applications [125, 147, 183], and embedded systems [123, 251]. This would allow one to affirm that test activities are highly dependent on the SUT. As a consequence of this dependence and particularity of the SUTs, many software companies allocate a specific team of practitioners to develop their own testing tools [22].

Despite particularities testing activities have generic points, as well. These general points are techniques and criteria that aim to mitigate efforts to discover errors and to decrease testing costs [142]. Testing techniques and criteria support the achievement of testing activities in a systematic and judicious way. Each technique has the support of various test criteria to perform the exercise of different features of the SUT [25, 169]. Therefore, testing techniques can be considered strategies to minimize effort and maximize efficiency for detecting defects [142]. Regarding only testing techniques, one can highlight mainly “Functional testing”, “Structural testing” and “Error-based testing” [10].

Conventional wisdom has it that Functional testing, also known as black-box testing, consists of a testing strategy based only on the program’s specifications [169]. This testing strategy disregards SUT’s internal structures and it can be carried out using a set of test cases. On the other hand, Structural testing, also called white-box testing, employs SUT’s internal structures as a source of information for different tests and coverage analysis. Finally, Error-based testing uses information about frequent errors that are common during the process of writing code in software development projects [10].

In addition, during software development processes there are testing phases to be considered. Among these phases the most popular is: unit testing, which is the phase where the main test focus is to assess the correctness of the smaller units of the SUT (e.g: functions and modules); integration testing, whose goal is to verify the adequacy of different software modules working together as whole solution; and system testing which aims at detecting faults in systems [25, 142, 169].

Applying appropriate software testing techniques and criteria during all phases of the development process imply a general increase of project costs for developers and, consequently, customers [142]. However, according to Seo e Choi [196], complexity and costs of software testing increase as do the size of the SUT. Consequently, at every stage
of the development life cycle, it is necessary to apply testing techniques and strategies. Thus, many times, testing activities take more than 50% of the time of development [169]. For instance, it is possible to mention that a proper test for an SUT as a whole would take enough time to delay official commitments and deadlines. Therefore, different software companies have a policy of releasing unfinished systems under development and waiting for complaints from users to improve the system’s quality and reliability [98, 218]. It is often said that software developers and testers, generally, do not have enough time for testing their projects.

The most appropriate solution to alleviate the problems of cost and time associated with software testing is using automated tests during different stages of the development process and in different modules of the SUT [26]. An adequate automated testing has to provide productive and reliable approaches [100]. Test automation is a matter of emphasis on the SE area because they promote more systematic testing approaches [26, 100]. Automated testing activities can bring significant cost savings to software development projects, and is a key factor to help reducing analysis costs [10]. Section 2.2 presents common practical aspects associated with the automated test.

2.2 Automated Software Testing

Due to the plurality of software systems available in the last years, it is possible to determine different meanings for the term automated testing. According to Dustin et al. [63], an automated test may mean a wide variety of aspects associated with the testing processes such as, using capture/record/playback tools, test-driven development, unit testing, custom scripts, etc. In spite of these variations, the authors highlight four fundamental characteristics that differs between manual and automated testing approaches: (1) automated tests are software development; (2) test automation enhances manual efforts once its focus is on automating hard manual tests; (3) automated testing activities do not replace manual tests and human intervention with testing strategies and techniques; and (4) automated tests cannot be separated from manual tests because these approaches are complementary.

To figure out an adequate definition of automated software testing, we prefer to list some basic conditions that should be achieved in a testing scenario. Some of these conditions were suggested by Hoffman [87]. According to this author, all conditions need not necessarily be satisfied to set an automated testing environment. These conditions are:

- Loading and running test cases automatically;
- Lacking human intervention after setting test cases;
- Allowing automatic system conditions setting and persistence of these conditions;
• Including resources for getting relevant results;
• Including resources for comparisons between current and expected results; and
• Including methods to analyze each individual test case result (pass/fail).

On the other hand, for the manual test, a human (the tester) provides inputs to the SUT and manually verifies its outputs. Manual testing involves six steps that do not necessarily need to be performed completely [93, 169]: (1) coding; (2) code update in some excerpts; (3) build/compilation; (4) running code and, sometimes, filling out test forms; (5) checking log files, databases, external services, screen outputs, variable’s values, etc; (6) tester/developer results checking. In this scenario, manual approaches have become a common practice among software developers before they deliver a version for a testing team. In case the code does not work as it is expected to, some previous steps should be reconsidered [26, 88].

Table 1 presents basic comparative characteristics regarding automated and manual testing environments. These characteristics should be fairly presented pondering important aspects of a development and test environment such as cost and productivity [88]. The advantages of adopting automated testing are worthy: simple replay, regression support\(^1\), time to market and high productivity are some of them [26, 83]. On the other hand, manual testing is simple to adopt and has a low cost in the short term [88].

Table 1: Implications for Manual and Automated Testing.

<table>
<thead>
<tr>
<th></th>
<th>manual</th>
<th>automated</th>
</tr>
</thead>
<tbody>
<tr>
<td>test conduction</td>
<td>human-based</td>
<td>machine-based</td>
</tr>
<tr>
<td>re-execution</td>
<td>time-consuming</td>
<td>quick</td>
</tr>
<tr>
<td>regression support</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>coding</td>
<td>simple</td>
<td>complex</td>
</tr>
<tr>
<td>project time</td>
<td>time-consuming</td>
<td>short</td>
</tr>
<tr>
<td>productivity</td>
<td>low</td>
<td>high</td>
</tr>
</tbody>
</table>

There are many deep and fair comparisons between manual and automated testing regarding variations in project and team size. Using this thinking as a premise, Table 2 draws a comparison where advantages and disadvantages are numbered. In this comparison we consider several aspects such as, complexity, efficiency, efficacy, productivity, and cost. For the most of these aspects, automated approaches work better than manual ones. Although it is true that human monitoring is necessary and imperative for a wide variety of test scenarios. The most notable advantage of manual approaches is the fact that a person running tests manually can identify unexpected behaviors that automated

\(^1\)Regression test should regard outputs from previous version of the SUT and use these outputs as expected outputs in recent versions of this same SUT.
testing might not be able to identify. A human being is able to notice screen oscillations, delays or any unexpected behavior that an automated testing system would have difficulty identifying [87, 88].

Table 2: Comparison Between Manual and Automated Testing.

<table>
<thead>
<tr>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>• a human being monitoring might notice</td>
<td>• efficiency in detecting errors must consider</td>
</tr>
<tr>
<td>unexpected behaviors</td>
<td>human fatigue</td>
</tr>
<tr>
<td>• notification of inconsistencies regarding</td>
<td>• determined test data, iterations and combinations may</td>
</tr>
<tr>
<td>different executions of data, states, settings</td>
<td>require automation</td>
</tr>
<tr>
<td>and environments</td>
<td>• certain aspects may not have manual solutions (i.e.</td>
</tr>
<tr>
<td></td>
<td>performance analysis)</td>
</tr>
<tr>
<td></td>
<td>• when testers are waiting for certain errors, after a few</td>
</tr>
<tr>
<td></td>
<td>repetitions, this may impairs the detection of defects</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>• more complex, efficient and well elaborated</td>
<td>• some complex domains of test provide</td>
</tr>
<tr>
<td>test cases</td>
<td>extreme difficulty for automations</td>
</tr>
<tr>
<td>• tools are able to match amount, productivity,</td>
<td>• only check the conditions for which the automation was</td>
</tr>
<tr>
<td>and efficiency</td>
<td>designed to investigate</td>
</tr>
<tr>
<td>• possibility of technology transfer between</td>
<td>• difficulties on defining which conditions should be checked</td>
</tr>
<tr>
<td>industry and academia</td>
<td>to consider a test as successful</td>
</tr>
<tr>
<td>• efficiency for error detections in specific</td>
<td>• implementations to generate efficient test</td>
</tr>
<tr>
<td>domains</td>
<td>cases are complex</td>
</tr>
<tr>
<td></td>
<td>• difficulty in predicting the way an SUT</td>
</tr>
<tr>
<td></td>
<td>should behave and what consequences are expected after an</td>
</tr>
<tr>
<td></td>
<td>execution</td>
</tr>
</tbody>
</table>

Over the last three decades, automated testing approaches have suffered from advancements. As a representative illustration of these advancements one can include the improvement of test oracles approaches. Year after year new studies provide more efficient and productive ways for SUTs have their outputs checked, aiming to reduce human efforts. Section 2.3 presents the main conceptual aspects associated with test oracles in automated testing environments.

### 2.3 Test Oracles

A plurality of definitions and concepts can be associated with test oracles. According to Machado [137] and Hamlet [81], the definition of test oracles concerns decision procedures and strategies for interpreting and judging the test results. When a set of test cases TC is available, the tester might implement test oracles as an object comparator between test case outputs and SUT’s current output. In this case, regarding an SUT P, one can
consider an input domain \( D \), an output domain \( X \) and, then, \( TC \) should be a function for matching the inputs \( D' \subseteq D \) and the outputs \( X' \in X \). In this scenario, \( D' \) corresponds to a valid input in the domain \( D \) and \( \text{dom}(TC) = D' \) denotes the domain of \( TC \). In this sense, \( TC(t) \) should be considered a valid output for each \( t \in \text{dom}(TC) \). Then, \( P(t) \) represents the result of \( P \) executing with input \( t \). Then, one can define that the program \( P \) meets \( TC \) if and only if it meets it on all inputs in \( \text{dom}(TC) \). Therefore, considering that \( TC \) might have infinite correspondences, a finite set of test cases should be selected. Considering this scenario, a function \( O \) called oracle for \( P \) on \( TC \) if for all \( t \in D \):

\[
O(t) = \begin{cases} 
\text{true (pass)} & P(t) = TC(t) \\
\text{false (fail)} & P(t) \neq TC(t) \\
\text{true (pass)} & t \notin \text{dom}(TC) 
\end{cases}
\]

Technically, test oracles represent a mechanism to check test outputs. One can consider this short definition as satisfactory, however real software testing scenarios and different test output domains provide ample evidence that this premise must be interpreted in different ways to be considered true [35, 57, 174]. To put it succinctly, before presenting more deep definitions, it is necessary to consider test oracles as a piece inside the software testing context. This piece can be represented, for example, by a tester checking outputs and writing test reports. In this case, in particular, the oracle is called a “human oracle”. On the other hand, this piece can be totally automated by several processes able to judge the correctness of test outputs.

Regarding the automated testing scenario presented in Section 2.2, one can observe that it is possible to set a test environment in which an automated oracle is not included. In this scenario, generally, testers apply several techniques for achieving structural coverage and in generating test data. However, the SUT’s outputs or behaviors should be checked or evaluated by a human tester (human oracle). The efforts and project time expended by the tester on evaluating test outputs is known as the “human oracle cost” [149]. Human oracles are common in several situations such as, evaluating the effectiveness of new test techniques, using the tester know-how about the SUT behavior, detecting unexpected errors, etc.

In order to achieve productive software testing in large scales, testers should automate their test oracles to mitigate manual efforts. Disregarding manual approaches, oracles play a vital role in automated testing and their neglect can result in an unproductive test. To illustrate this, Figure 1 presents a wide concept map of software testing context and it illustrates test oracles as a supportive technology [61]. The concept map represents a general view of software testing matching sources of information, test phases, test techniques, testing criteria, processes, and test oracles. According to this figure, a test
oracle is a software testing technology that can be associated with different processes and test techniques.

Figure 1: A Concept Map of Software Testing. (Extracted from Durelli et al. [61] with kind permission from the authors)

Figure 1 provides a concept from which it is possible to infer that test oracles are independent from any other software testing aspects. Regarding a more specific software test environment, test oracles play a fundamental role as the conclusion of a series of processes that might include test case generation, test data selection, test sequencing, and test execution. In order to illustrate a scenario like this, Machado [137] suggested a generic model of a black-box testing process including test case generation, test data selection, automated execution, and test oracle. Figure 2 provides a practical example of a generic association among all of these phases in a practical scenario.

Figure 2: A Generic Model of a Black-box Testing Process.

In Figure 2, depending on the output domain, a tester can implement a productive test oracle in the form of a program, a set of assertions, a function, a heuristic or a data structure with expected outputs. Addressing these varieties, according to Chan & Tse [36], under practical and theoretical points of view, test oracles can frame fundamental
Concerning the theory, Chan e Tse [36] argue by means of the program’s specifications one is able to determine appropriate test oracles. However, it is important to highlight that, in practice, oracle mechanisms could be costly, hard to implement or may not exist [16, 83, 235]. So, the implementation of an automated mechanism to the same end as a human being judging test outputs is a challenging task to be faced by test designers [216].

Despite the variety, in practice, aspects related to test oracles deal with a primary issue in software testing activities – deciding about the correctness of an SUT \( P \), against pre-determined test data. Figure 3 presents a traditional and generic test oracle structure using flowchart elements. In this case, the test oracle accesses the set of data needed to evaluate the correctness of the test output. This set of data comes from the specification of the SUT and contains sufficient information for supporting the oracle’s final decision.

**Figure 3: Generic Test Oracle Structure.**

Figure 4 introduces several structures which represent test oracle functions. In the figure it is possible to notice that, in order to adequately play their rules, some test oracles may require test cases. However, there are cases where test oracles are able to provide a test result based on test data (inputs). There are cases where a set of information and data about the expected results are needed to decide about the correctness of SUTs (Figure 4a) [78, 89, 103, 114, 148, 214]. On the other hand, test designers can create test oracles from formal methods or specifications (Figure 4b) [6, 12, 41, 60, 73]. Regarding most sophisticated cases, test oracles can place into service test data inputs to derive expected outputs of the SUT (Figure 4c) [187, 193, 198, 199, 228]. Finally, as we already mentioned, one can consider the human oracles, where testers can use their own knowledge about the SUT to check if outputs are in accordance with specifications (Figure 4d) [2, 90, 149, 209]. Other possibilities could be contemplated, for example, a test oracle can be designed considering a reliable model of output from which basic features can be extracted to be compared with current outputs from the SUT [57, 174].

Regarding complexity and costs, one can consider test oracles based on expected output/behaviors as unsophisticated and quick to implement. Regarding Figure 4a, a tester can deal with these oracles using one of the various frameworks that have come
(a) Using Expected Output/Behaviors.  

(b) Using Formal Models/Specifications.

(c) Using Test Data.  

(d) Human Oracles.

Figure 4: Test Oracles Using Different Sources of Information.

to be known collectively as an “xUnit” family. xUnit frameworks allow the unit testing in SUTs implemented in different programming languages [104]. For example, in “JUnit” [222], which is an xUnit framework for Java, testers can get different test oracles in their code using assertions, which mean a true/false statement placed in a program to check unit and partial results. Figure 5 presents an example of a test oracle using the JUnit framework. In this example, line of code 4 represents a test oracle to check the correctness of the method “boolean Store.checkByTitle(String)”.

```java
1  public void testDVDInStore ( ) {
2   Store store = new Store ( ) ;
3   boolean result = store.checkByTitle ("The Godfather III") ;
4   assertEquals (true , result ) ;
5  }
```

Figure 5: Test Oracle Using an “xUnit” Framework.

According to Figure 4b, when testers have at their disposal a mathematical model that faithfully represents the SUT such as, a Finite-State Machine (FSM) or a Petri net, it is possible to automate a test oracle from this model. Along the same lines of the example presented, regarding to Figure 4c, it is possible to set a test oracle using test data. For instance, through outputs given from parallel executions or using another versions of the SUT, a tester can build a test oracle to compare those outputs and current outputs. In these cases it is important to assume that the version used, called the reference program, meets all the specifications of the SUT. This sort of test oracle frequently uses the cases of a regression test and a mutation test.

2.3.1 The Oracle Problem

The decision about the correctness of an execution and, consequently, revelations of failures is the most essential aspect related to any testing activity, even the manual ones.
In a software testing environment, test oracles play this essential aspect, which is a hard-task because it is not trivial to find out a set of expected outputs for SUTs. For instance, we can wonder about a situation in which the SUT P must figure out the value of π on a determined precision of decimal numbers. Unless it has another program Q, which is working properly to solve the same problem, it is impossible to determine if P’s outputs are right or not. Another similar case is when P represents a non-deterministic program and P(vk) matches different possible outputs, all of them correct. That means, the process of deciding about correctness of SUT’s is often much harder than it seems and involves finding solutions for different kind of problems. Problems like these have been frequent in the software industry during the last three decades. Due to this, testers consider the task to determine reliable sources to judge all SUT’s outputs a complex and non-trivial.

The “oracle problem” is set in cases when, using practical means, it is impossible or too difficult to decide about the correctness of test cases and test outputs. Along the same lines, we would define the oracle problem as the absence of oracle or cases when it is too expensive to apply the oracle. There are not enough test oracles to always support the right decisions. Due to this fact, the oracle problem is a fundamental challenge presented in the literature often. This is one reason why testers classify the design of test oracle as a complex and cognitive activity rather than being a routine activity. Even having a test oracle to support evaluations of test results, practitioners frequently identify unpredictable points about the use of oracles. Depending on the oracle, the following problems may occur:

- **False positives**: the test result passes, however, some inconsistent state might not be checked. Regarding a test scenario, a false positive is when a test result incorrectly rejects a true null hypothesis; and

- **False negatives**: cases in which the test result fails while the SUT is working well. In a test scenario, a false negative is the failure to reject a false null hypothesis.

Some research aim to alleviate the oracle problem using specific techniques. In these cases testing tools that are able to support testers’ decisions about the correctness of an SUT execution. However, testers must also consider the possibility that oracles can often support them making wrong decisions. Depending on the SUT, it is extremely difficult to predict behaviors to be compared against current behaviors. Failures can come out under different situations which make checking the result complex or impossible to be performed. The oracle problem can be challenging when SUT outputs are given in complex formats such as, images, sounds or virtual environments.
2.3.2 Trade-off on Test Oracles

Dealing with test oracles brings some trade-offs. For testers, these situations involve losing project time and increasing costs in return for gaining quality aspects. Scientific studies explore the trade-off between efficiency in identifying failures in SUTs and complexity in developing oracles [145, 190, 257]. The more effective the oracle, the more complex the information that it uses to determine if the program execution is in line with the expected results. Consequently, defining an oracle as complex as the original program is usually not acceptable.

Other trade-offs must be explored as well. For instance, if it is possible to define a fully automated oracle to identify all possible failures present in the SUT’s outputs, there would be no need for the program itself, because the oracle could be used instead. Then, we could consider that, regardless of the goal in studying a new oracle approach, it is important to know: What already exists? Where does the new approach fit? And how do we compare them? This allows, for example, one to identify what position in the trade-off a new study fits. For practitioners, it becomes easier to identify what kind of approach is best suited for a particular problem, that is, which oracle approach should best be used to support the test of a software product.

Considering broader scenarios of test automation including test oracles, testers should anticipate dealing with more complex trade-off situations. In these cases, the trade-offs increase because an important aspect is that both the reliable test set problem [91] and the oracle problem [235] are open issues. A set of test data $T$ for a software system $P$ is reliable if it reveals that $P$ contains an error whenever $P$ is incorrect. According to Howden [91], an effective testing strategy which is reliable for all software systems cannot be implemented. In this context, there is often no way to define in practice a criterion test that, once applied, can identify all errors in every program; likewise, there is no way to define in practice a test oracle that identifies all wrong outputs in every program [55]. In particular cases, this means that one must consider existing resources, the number of possible inputs (input domain), complexity in defining the respective outputs for all inputs, time, and cost. Thus, the tester must always weigh the resources available with time and test effectiveness.

A lot of research on software testing focuses on approaches that seek to change the balance of the trade-off in favor of some variable related to it. In the case of reliable test set problems, for example, many studies aim to find an approach or data selection criteria to identify more errors with a fixed and finite amount of resources for an application domain [213, 214]. Other areas of research involve the generalization of results, i.e., whether an approach for a specific domain can be used in other domains, and how effective it is [225, 229]. Comparative studies are often published to answer such questions [4, 197].
Section 3 brings explanations about the most studied and reported test oracle taxonomies in accordance with researchers and practitioners.

3 Oracles Taxonomies

Different authors [22, 24, 83, 86–88, 216] have dedicated their efforts in writing theories and classifications and taxonomies of test oracles. Due to the diversity of software domains, currently there is no standardized test oracle taxonomy and the existing classifications could depend on the source of information, the SUT’s output or the automation method considered by the oracle. In this section we present some of the most common oracle taxonomies and classifications. We present oracle taxonomies found in different kinds of SE studies conducted by academia members and practitioners. In this section we separated these classifications into three groups: (1) generic taxonomies; (2) specific taxonomies; and (3) taxonomy using oracle information characteristics. In addition, we present some example to support theoretical descriptions.

3.1 Generic Classifications

In this section we introduce general classifications in which test oracles can or cannot fit. Although these classifications are well known by many researchers and practitioners, they are not often mentioned in studies of the test oracle. These classifications are associated with the general purpose of the test oracles and they are not associated with technical aspects applied in order to implement the oracle function such as, source of information or automation level. Sections 3.1.1 and 3.1.2 present some characteristics that oracles can or cannot meet.

3.1.1 Pseudo-oracles and Partial Oracles (Davis e Weyuker [55] and Weyuker [235])

Davis e Weyuker [55] were two of the first authors to formalize and discuss the oracle problem. They sought solutions that alleviate the fact that it is impossible in practice to define a complete and totally reliable oracle for all SUTs. Given the impossibility of defining an ideal oracle, the authors discuss two possible options: “pseudo-oracles” and “partial oracles”.

According to Davis e Weyuker [55], pseudo-oracles are programs written by a second team, in parallel to the SUT, and following the same specifications. Both, oracle and SUT run with the same input data and outputs are compared. This concept can be considered in two aspects. First, one could consider any code or executable model as a pseudo-oracle,
Second, the margin of accuracy between the pseudo-oracle and SUT could be applied to allow acceptable discrepancies between results.

There is no guarantee that an oracle is free of faults hence “pseudo”. Accordingly, when the SUT’s output is not equivalent to the oracle output, one must go through the debug process to check which one actually has the fault. As we mentioned before, if the oracle is incorrect, i.e., the oracle indicates a failure when there is no failure, this is a false negative. On the other hand, if both programs present the same wrong result, the verdict will be that the SUT passed the test when it actually did not – false positive.

Along the same lines, partial oracles aim to identify where test results are incorrect, even without knowledge of the correct output. In other words, a partial oracle should support the tester’s decision about the correctness of a test execution without expected outputs. Testers might resort to partial oracles when it is impossible or too difficult to define the expected result. That is, they must account for what is certainly wrong and analyze whether the SUT’s output is plausible. For instance, one can affirm that a result of a \( \sin \) function cannot be outside the “-1” and “1” boundary. If it is outside such a range, the result is clearly wrong. Examples of partial oracles are found in many different ways. Pre- and post-conditions often define contracts which must be respected in the code. Although it is true that if an SUT passes a test and no error is found, it may still be incorrect because the partial oracle only indicates that the program produces data within a plausible result, but it does not mean the result is confidently correct.

Partial oracles may seem impractical or inefficient in the present day, as it was presented in 1981 and it is natural to think that nowadays larger and more complex programs make it impractical to implement parallel and costly versions of the SUT. However, a literature review shows that pseudo-oracles can still be viable and studies are still in use. For instance, 24 years after Davis e Weyuker’s proposal, Hummel e Atkinson exploit technologies to find components for reuse as a means of discovering pseudo-oracles on the Internet. In this case, the components found have the same function of an SUT and can be used as pseudo-oracles to support statistical testing of a self-built component implementation. This idea is explored further in the present day by other research, as with the use of web-services.

3.1.2 Passive and Active Oracles

Along the same lines of the classification presented before, the literature presents a generic classification among test oracles: passive and active oracles. It is often said that active oracles directly drive their own testing activities reproducing the SUT behavior and generating expected outputs. Contrary to this premise, passive oracles act as simple comparators between current and expected test results.
et al. [178], an active oracle mimics the behavior of the SUT and a passive oracle verifies the SUT behavior, but it does not reproduce it.

Active test oracles are common in the literature. A traditional example is test oracles using formal models to reproduce expected outputs. On the other hand, as an example of a passive test oracle, one can mention the approach present by McDonald et Strooper [145], in which passive test oracles are implemented from translations of Object-Z specifications. Another example is provided in the study conducted by Shukla et al. [206]. In this approach, the authors present a passive oracle built as a wrapper with checking functions based on the API (*Application Program Interface*) of a software component. According to the authors, this technique can be applied in a wide variety of software components in most programming languages.

The active-passive oracle concept is transverse to the pseudo and partial oracles (Section 3.1.1). A program written in parallel, i.e. a pseudo-oracle, reproduces the behavior of the SUT, therefore, it is also an active oracle. The partial-oracle which does not reproduce the result but compares it with a set of constraints is a passive oracle.

### 3.2 Specific Taxonomies

The general purpose of this section describes specific test oracle taxonomies on software testing presented in the literature. Research on test oracles defined the majority of these classifications regarding different aspects such as source of information, automation process, information characteristics, etc. In this section we present these classifications highlighting their particularities. Each subsection represents a particular classification and the context in which this classification was suggested. We use a broad variety of practical examples to support our explanations.

#### 3.2.1 Source of Oracles (Beizer [24])

Beizer [24], interested in characteristics used as sources of information in different test oracles, identified five different types of oracles. According to him, oracles are defined following different aspects such as, source of information, data input, generation of expected outputs, etc. Beizer [24] then defined five types of oracles according to their sources of information:

- *Input/outcome oracle*: more complex and the most common, this test oracle specifies the expected outcome for a specified input;

- *Kiddie testing*: the principle of these oracles is common; it runs the test using test data and reporting whether the application crashes. In more complex cases, this
type of oracle should verify some characteristics searching for incoherency, and after this, it gives a verdict;

- **Regression test**: expected outputs are given from the previous version of the SUT. Besides having a new version of the SUT, the tester should consider test data with the same outcome as the last version;

- **Pattern test set**: decisions are based on a pattern previously created and validated. This type of oracle is common for validating compilers, Web browsers, and word processors; and

- **Existing programs**: the test is performed based on the results of entries executed on an existing and similar system.

### 3.2.2 Classification by Automation Process (Hoffman [86, 87, 88])

According to Hoffman [87], besides serving as a reference for comparison of current outputs with expected results, the expression test oracle could be used to describe the generation of expected results (data). Then, disregarding manual approaches, behind every test oracle exists an automation process. In three different studies [86–88], the researcher Douglas Hoffman described several characteristics useful to the process of oracle automation. In the same studies, Hoffman presents a different oracle taxonomy. In this context, we compiled these characteristics defining a taxonomy based on the automation processes described by Hoffman. This taxonomy is presented in Table 3. The table shows the oracle classification, expressive features, advantages, and disadvantages classified into different classes.

### 3.2.3 Oracle Categories According to Harman et al. [83]

In a recent scientific study, Harman et al. [83] provide a broad analysis of trends in research on test oracles regarding current approaches. In their analysis, the authors considered the works on test oracles into four different categories:

- **Specified oracles**: in this category Harman et al. [83] include all test oracle that specifications in order to judge test outputs and behaviors. Naturally, a lot of different studies could be included in this category. Among these studies, the authors highlight specification based languages, models, assertions, contracts, and algebraic specification;

- **Derived oracles**: in this group of test oracles, tests included approaches using any type of artifacts or resources from which test oracles may be created. Given this
Table 3: Compilation of Testing Oracles by Their Automation Process.

<table>
<thead>
<tr>
<th>Oracles</th>
<th>Definition</th>
<th>Advantage</th>
<th>Disadvantage</th>
</tr>
</thead>
<tbody>
<tr>
<td>No oracle</td>
<td>Approach without verification of results</td>
<td>Large amount of data can be executed</td>
<td>Only notable errors are detected</td>
</tr>
<tr>
<td>Human</td>
<td>A human figure verify results</td>
<td>Can detect unpredictable errors</td>
<td>Time consuming and its efficiency is influenced by physiological factors, such as, tiredness</td>
</tr>
<tr>
<td>True or Complete</td>
<td>The oracle generates outputs from inputs</td>
<td>All possible errors, considering the test set, are detected</td>
<td>Expensive implementation. Its execution is time consuming</td>
</tr>
<tr>
<td>Consistency</td>
<td>It compares current results to previous results (regression testing)</td>
<td>Quick verification. Can verify large amount of data</td>
<td>Does not detect errors in the original program</td>
</tr>
<tr>
<td>Stochastic or Random</td>
<td>Checks a random test data sample</td>
<td>Could automate testing in a simple manner</td>
<td>May not notice specific errors and it is time consuming to check</td>
</tr>
<tr>
<td>Sampling</td>
<td>Verifies a previously selected test data sample</td>
<td>Generally promotes a quick check</td>
<td>May not detect specific errors</td>
</tr>
<tr>
<td>self-referenced</td>
<td>Promotes responses to data via messaging</td>
<td>Allows wide post-test analysis. The verdict is given in the message content</td>
<td>The system must submit test results so that messages are received</td>
</tr>
<tr>
<td>Heuristic</td>
<td>Verify specific characteristics</td>
<td>Quick, simple and inexpensive</td>
<td>Generally could bring false positive/negative results</td>
</tr>
<tr>
<td>Model-based SUT</td>
<td>Uses a digital model data about the behavior of the SUT</td>
<td>Tests can be used across multiple systems using different models</td>
<td>Maintenance is expensive. Complex models must match the expected behavior</td>
</tr>
<tr>
<td>Manual</td>
<td>Results should be carefully defined by a test engineer</td>
<td>Useful for complex SUTs</td>
<td>It always performs the same procedures and is limited by a number of test cases</td>
</tr>
<tr>
<td>Statistics</td>
<td>Makes statistical correlations between inputs and outputs</td>
<td>Enables checking large scale and real data systems</td>
<td>May not detect apparent errors</td>
</tr>
<tr>
<td>Computational</td>
<td>Explores the behavior of the SUT to convert inputs into results</td>
<td>Very useful for simple mathematical functions and transformations</td>
<td>May not detect some errors and generate false positive/negative results</td>
</tr>
</tbody>
</table>

Definition, the authors highlight that derived oracles may become an uncompleted specified oracle. The common resources cited are MR, regression test suites, system executions (trace), textual documentation, and strategies for pseudo-oracles;

- *Implicit oracles:* are intended to identify situations in which the presence of a fault in the SUT is obvious such as, crashes. Harman et al. [83] highlight that implementing these oracles requires no domain knowledge and the technique can be used in all
runtime problems. A function which is always waiting for handles of exceptions is an example of this test oracle;

- **No oracles**: in this group of study, the authors include some approaches on test oracles which try to deal with the test oracle as a whole. In these studies, despite the absence of automated oracles, the researchers’ target is generally to reduce the human efforts on the judgment of test outputs. Among these targets, the authors highlight reducing human oracle cost, reducing qualitative human oracle cost, and crowdsourcing the oracle.

### 3.3 A Taxonomy by Oracle Information Characteristics

Oracle information is a standardized term by researchers of test oracle use in order to reference data about expected test outputs [172]. Then, oracle information represents the SUT’s expected behavior [151], which might be obtained from the specification, stored results, parallel program execution, learning machines, and other sources. The oracle information can be concrete (that is, the expected result itself), or abstract, as the acceptable boundary of results, expressed by a constraint. Complementing the concept of oracle information, oracle procedure is another standardized term used by researchers and it represents the processes used to compare the oracle information with the current output. According to Durrieu et al. [62], systems perform this comparison at runtime (online) or after the execution (offline).

Given this introductory scenario, the source of information and the way this information is represented, can influence the accuracy, complexity, expressiveness and many other aspects of the oracle’s effectiveness. Furthermore, the sources of information are critical components to determine the ease of writing and interpreting their meaning. In this section, we suggest an oracle classification regarding the oracle information explored by studies on test oracles. This taxonomy assumes that studies on test oracles fit into four basic types: (1) specification-based oracles, (2) metamorphic relation-based oracles, (3) machine learning oracles, and (4) version-based oracles. Disregarding a few differences, our taxonomy is quite similar to classification suggested by Harman et al. [83]. It is important to highlight that this classification is used in the mapping presented in Section 4 of this chapter. So, this section presents this taxonomy in more detail and with practical examples.

#### 3.3.1 Specification-based Oracles (Baresi e Young [22])

Baresi e Young [22] present a specific survey highlighting the main characteristics of automated test oracles that require neither pre-computed input/output pairs nor a previous
version of the SUT. Consequently, regardless pre-computed outputs, software specifications are the most important source of information to derive test inputs and test outputs. This category of test oracle can be named specification-based. Due to the popularity of this test oracle approach, this section brings several examples of differences forms to implement test oracles using SUT’s specifications.

Technically, a specification can be defined as a detailed formulation, in document form, which provides a definitive description of a system for the purpose of developing or validating the system [97]. When this formulation is used for validation, it is regarded as the oracle information. The language used to describe such a formulation is called a specification language, which is capable of being interpreted by a compiler or interpreter and allows the creation of an automated oracle procedure.

The interest in using specification languages as oracle information is also supported by a variety of studies found in the literature (Section 4). Specification-based oracles can be classified according to the paradigm, automation features, levels of abstraction, and other characteristics. Here, we present specification-based oracles regarding three general aspects: (1) their location in relation to the SUT, (2) their paradigm, and (3) their temporal property representation.

**Specification Location:** Baresi e Young [22] present groups of oracles according to certain similarities, among them: oracles of pure specification language, embedded assertion languages and extrinsic interface contracts. These three oracle contracts differ from each other in the way they are written with respect to the code.

Oracles based on “pure specification” are those in which the tester uses a specification language to describe the desired behavior of a system or part of it for later use as a source for the oracle, i.e., as the oracle information. Such languages are usually not designed with the concern of being automatically interpreted. Therefore, defining procedures to interpret oracles which use such a source of information can be challenging.

“Embedded assertion languages” allow one to insert expressions of intent within the code to be tested. They usually represent pre- or post-conditions at some control point in the program. Such expressions are checked during the code execution and, if a violation is detected, a message is presented. In this way, they are executable code which may be an extension of the same language used to implement the SUT or from a different language.

Java and other programming languages natively support embedded assertions. There are also specification notation languages, such as Anna [80, 134, 194] and JML [49, 164], in which the assertions can be written in the code. In such cases, assertions are marked with

---

2Assertion may be defined as a logical expression specifying a program state that must exist or a set of conditions that program variables must satisfy at a particular point during program execution [97].
reserved words recognized by an outside interpreter/compiler and executed as a separated part of the code.

For example, consider a Java code snippet which was modified to increase its performance, where the tester wants to be sure that the output is the same as its original output. In this context, one may introduce an assertion in the SUT to ensure that an error will occur if the refactoring method for coordinate calculation differs from its original. As an illustration, in Figure 6, the first line contains the embedded assertion recognized by the Java compiler and, consequently, JVM (Java Virtual Machine) given the reserved word “assert”. During the program execution, if the expression is not true, a runtime error is thrown. Technically, this is the oracle strategy explored by JUnit (Figure 5, Section 2.3).

```java
1    assert (calculate_coord() = newCalculate_Coord());
2    float new_coordinate = newCalculate_Coord(coordinate);
```

Figure 6: Excerpt from an Assertion Code.

“Extrinsic Interface Contracts” keep the specification, in the form of assertions, separate from the implementation and less closely tied to the target programming language. There are several assertion languages that allow extrinsic interface contracts, known as ADL (Architecture Description Language). Developers must define the bindings between the specification and the functions in the program [53, 82]. This is usually achieved by the use of wrappers. A wrapper is a checker that surrounds the component under test without modifying its code. For instance, when testing a given class on an OO code, another class (the wrapper) is created with the same interface as the original but with other methods which are responsible to check some constraint.

Figure 7 presents a code example, adapted from Shukla et al. [207], in which the objective is testing a class which is a list of integers, using an insert method. The code represents the list class to be tested. A list of integers, in this program, should have a maximum length of 1,000 elements and three methods: insert(), size() and exists(). To test the insert() method, two basic assumptions are considered: (i) if a value is inserted, it must exist in the list; and (ii) if a value is inserted, the list must have one more element than before the value is inserted. Both assumptions can be tested without inserting intrusive code (embedded assertions) with a wrapper.

Figure 8 represents such a wrapper. It extends the original class and overrides the method to be tested. The new insert() method (i) retrieves the size of the list before the insertion, (ii) calls the method under test, (iii) retrieves the size of the list after the insertion and (iv) calls the method which will evaluate the assumptions. If the value is not inserted into the list or if the list does not contain one more element after an insertion,
a message is sent. A driver calls the method `insert()` from the wrapper and the class under test is evaluated without the need for intrusive code.

```java
public class Wrapper extends ListOfInteger{
    public void insert(int value){
        int before = size();
        super.insert(value);
        int after = size();
        checkInsert(value, before, after);
    }

    void checkInsert(int value, int before, int after) {
        if (((!super.exists(value)) || ((before+1) != after))
            System.out.println("Error on insert() ");
    }

    public final int MAX_LENGTH=1000;
    private int index=0;
    public int [] list = new int[MAX_LENGTH];
    public void insert(int value){
        //The code insertion code is here
    }

    public int size(){
        return index;
    }

    public boolean exists(int value){
        //the code to check if an element exists is here
    }
}
```

Figure 8: Code Representing Class to be Tested.

Other specification languages may be related to the SUT by some variation of the presented approaches. For example, a specification does not need to be a contract, but can be any other language paradigm which can be translated by the oracle procedure. The mapping between oracle information and SUT can also be achieved by instrumenting the SUT to dump the target outputs to a log which can then be used by the oracle procedure in the analysis.

**Specification Paradigm:** There are several specification languages and their respective paradigms are reflected in many different oracle approaches. Because any programming language can actually be interpreted as oracle information, there are as many oracle information paradigms as language paradigms. Here we present some of the most common paradigms that can be found in the literature to write oracle information.

Embedded assertions, discussed in the last section, are “declarative languages” that are nonprocedural languages that permit the user to declare a set of facts and to express queries or problems that use these facts [97]. Other examples of declarative languages are
OCL (*Object Constraint Language*) and Alloy, which are not embedded assertions. Such oracles are examples of partial oracles.

OCL, for example, is an extension of UML (*Unified Modeling Language*) proposed by the OMG (*Object Management Group*). It allows the definition of constraint limits of values to variables and pre- and post-conditions of methods [30, 31, 48, 176]. An example of a study of automation is presented by Cheon e Avila [48], in which OCL constraints are translated to runtime checks in AspectJ, separated from the implementation code, and mapped into the SUT with pointcuts.

Other languages provide “procedural” or “object-oriented” resources, such as Object Z which allows the tester to plan his oracle information with such concepts as inheritance. An external checker is needed to translate specification sentences, as previously stated. Perfect Developer [52] is a tool that incorporates an automated theorem to verify that specifications written with Object Z (or a specification language called Perfect) are sound. Such a tool is not an oracle in the sense that it does not compare the specification with the SUT. Rather, it analyzes whether the specification has flows that work as contradictions.

Another specification paradigm encompasses “executable models” which can be simulated in tools such as Simulink, XCos, and Scicos. They are suitable as oracle information because it is possible to simulate the SUT behavior. An oracle procedure should retrieve the SUT results and compare them with the model results. Such an oracle is a simplistic example of pseudo-oracles because it uses another version of the SUT to test it – the executable model [170].

A practical example of the use of executable models as oracles is presented by Lasalle et al. [122]. The authors apply a Simulink model as oracle information. Its simulation calculates the expected results of a vehicle reaction with respect to the road characteristics when steering.

State machines and other models can also be used as oracle specifications. A parser can be applied to generate an analyzer from the machine. The SUT’s output is inserted into the analyzer. After that, if the output is not expected in the machine, an error is detected. In this case, the state machine plays the role of the oracle information and the analyzer represents the oracle procedure.

Andrews e Zhang [12] give an example of state machines as oracles to test an elevator system. In their model, the output is stored into a log file. The requirements are: (i) the door must be closed if the elevator is moving; (ii) the elevator must be stopped when the controller program terminates; and (iii) the door must never be open for more than 30 seconds. A sample of SUT’s output is presented in Figure 9.

AspectJ is an extension of a program paradigm known by *Aspect-Oriented Programming* (AOP) for the Java programming language.
Figure 9: A Sample of SUT’s output Regarding State Machines as Oracle.

The testers have created two machines: one for requirements 1 and 2 (Figure 10 (door safe)), and another for requirement 3 (Figure 10 (no delay)).

![State Machines Diagram]

Figure 10: State-Machines as Test Oracles (Adapted from Andrews e Zhang [12])

An analyzer receives the following inputs: the state machine and the log file. If the analyzer cannot recognize a line from the log or if there is no transition in the machine, the file is rejected and an error is identified. Eventually, the analyzer may be modified to ignore valid log lines that are not represented in the state machines. In the example, “call” and “reach” are expected lines in the log file, but they are not represented in the state machines. In this case, they must be ignored by the analyzer so as to not trigger an error.

**Temporal Specifications:** Some software systems must be in agreement with requirements that express temporal properties. This is common for embedded systems, as in Avionics and Telco. A requirement example of a Mars probe [234] is expressed as follows:

“... When the TimerInt reaches the Control System and the reading of acceleration is not completed, the status should change to Emergency within 60 milliseconds ...”

One can observe that a behavior (status changing) must be true within 60 milliseconds, that is, not necessarily at the same instant as the other triggering behaviors. The oracle
must have the ability to check such requirements in the SUT. There are many specification languages from diverse paradigms which can be used to write temporal properties. TRIO [68], MITL [234], EAGLE, GIL, time Petri nets [128], and timed automata [234] are other instances.

Higher expressive temporal languages should express time quantitatively, qualitatively and provide some representation of existential (∃) and universal (∀) quantifiers. Quantitative operators allow the representation of intervals of time in relation to other events. For example: “event X must hold until event Y happens”. Qualitative operators provide ways to measure distances, as “X must hold k units of time from the current instant”. For instance, Felder e Morzenti [68] provide such operators.

As with pure specification language, the execution of a temporal language is not always the main concern. There are studies which tackle the adaptation of pure languages on automated testing tools which are capable of translation. Given the high complexity of such languages and the difficulty of defining a complete solution, the adaptation is usually accomplished partially in exchange of some expressiveness.

This is the case with the research on oracles for Simulink-like models [170], where a tool is capable of analyzing results with respect to a TRIO-adapted specification (TRIO/Apolom). Although a recursive capability is not implemented, the oracle procedure is capable of analyzing quantitative and qualitative operators, such as existential and universal quantifiers.

The following requirements are used as a base to write an oracle information example in TRIO/Apolom language:

“... If the temperature of a system is greater than x for an interval of time (interval) of more than t1 instants, then a red indicator must be on to indicate a critical situation until it lowers to a safer temperature. A safety protocol (safety) must be started within the next t2 instants and should last the same number of instants as interval. Also, a yellow indicator must be on after the safe temperature is reached and until the alarm is turned off ...”

The requirement above can be translated to TRIO/Apolom as presented in Figure 11.

\[
\begin{align*}
\text{Starts}(\text{greaterthan}(\text{temp};x)) & \land \text{interval} = \text{NowOn}(\text{greaterthan}(\text{temp};x)) > t1 \\
\text{Lasts}(\text{on}(\text{red}), \text{interval} - 1) & \land \exists (\text{Lasts}(\text{on}(\text{safety}), \text{interval} - 1), 0, t2) \land \\
\text{UntilW}(\text{on}(\text{yellow}), \text{instant}() + \text{interval}, \text{off}(\text{alarm}))
\end{align*}
\]

Figure 11: Example of a TRIO/Apolom Translation.

According to Nardi [170], the left part of connector \( \land \) in the first line identifies the first instant (the starting point) when a temperature is greater than \( x \). The right side of
the connector has an operator to count how many instants the temperature is above \( x \) from the starting point and verifies whether it endures longer than is allowed \((t1)\). If both conditions hold, then the second line guarantees that the red indicator is on for the same interval of time as the critical temperature. It also verifies whether the safety protocol is activated for the next \( t2 \) instants and that it endures for the same interval of time as the interval. Line three insures that the yellow indicator is turned on the next instant that the temperature is below the critical point until the alarm is switched off.

### 3.3.2 Metamorphic Relation-based Oracles

The oracle information may not be based directly on the system specification, but rather on known relationships between multiple inputs and outputs, known as Metamorphic Relations (MR). MR specifies how the output of the program should change according to a specific change made to the input and represent some necessary properties of the \[45\]. MRs are used as test oracles in Metamorphic testing to identify faults in the SUT \[46\]. Following are the basic steps in applying metamorphic testing:

1. Identify set of MRs that should be satisfied by the SUT;
2. Create set of test cases using a traditional test selection approach such as random testing or fault based testing. These test cases are referred as initial test cases or source test cases;
3. Create follow-up test cases by applying the input transformations specified by the MRs identified in step 1 to the initial test cases created in step 2; and
4. Execute the initial and follow-up test case pairs to check whether the output change at runtime complies with the output change specified by the MR.

As an illustration, the \( \sin \) function must always obey the following property: \( \sin(x) = \sin(180 - x) \). Such necessary properties of the \( \sin \) function may not be a part of its specification. However, this property specifies a relationship between a pair of input data, \( x \) and \( 180-x \), and their respective outputs \( \sin(x) \) and \( \sin(180 - x) \). In the given example, the follow-up test case can be created by subtracting 180 from the initial input. Testing of an implementation of \( \sin \) can be performed by executing the following initial and follow-up test case pairs and checking whether the produced output pairs are equal:

\[ (3, 177), (15, 165), (45, 135), (125, 55), (277, -97) \]

Although this is a trivial example, MR based oracles are applied for testing applications in different domains, including bioinformatics \[43\], machine learning programs \[165, 242\].
embedded software [121], healthcare simulations [166], Monte Carlo modeling [58], and programs with partial differential equations [42]. In addition, [244] integrated MRs with program slicing and used that for fault localization in programs that face the oracle problem.

The set of MRs used in testing has a big impact on the effectiveness of metamorphic testing. Liu et al. [129] found that using more diverse MRs can improve the effectiveness of testing [129]. Usually, MRs used for testing are identified manually by the tester, based on her knowledge of the program specification. Recently, there have been developments of automated methods that use machine learning techniques to detect MRs [113]. These approaches use a set of features that represents the static control flow information of a program to develop machine learning prediction models that are used to predict MRs.

Some taxonomies on test oracles consider metamorphic relation-based oracles as a subcategory of derived oracles or specification-based oracles [83]. For the context of this chapter, due to the increase of approaches on metamorphic testing over the years, we have considered metamorphic approaches as their own category. We think that studies on metamorphic testing are mature enough to be analyzed and discussed separately.

3.3.3 Machine Learning Based Oracles

Machine Learning (ML) is a set of computational methods that use collected data and is capable of making predictions. There are a variety of ML techniques applied as oracles, such as Supervised Learning Machines (SLA): (1) Artificial Neural Networks (ANN) [5, 9, 108, 141], (2) support vector machines [232, 233], and (3) info-fuzzy networks [41, 252].

These three SLAs share a characteristic: they use collected and labeled data to train a machine to predict new, unlabeled data. ANN, for example, can be trained with pairs of inputs and already known outputs (training set). The resulting machine may be seen as an approximated function of the SUT. If a new input is used in the trained ANN it is capable of predicting an approximated output, even if this input-output pair is not part of the training set. Therefore, the ANN can be used as an oracle. If the same input is used in the ANN and the SUT, a procedure can compare both and identify if there are irregular discrepancies.

A similar application of ML as oracles is the use of them as classifiers. In the same way, pairs of elements and the class or category in which they belong are used to train an ANN. At the end of the training step, the ANN may be capable of identifying in which category new elements belong. For instance, Aggarwal et al. [5] and Jin et al. [108] present a case study with oracles for triangle classification into isosceles, scalene, equilateral or not a triangle. The ANN receives two inputs: a triplet which represents three sides of
a triangle and the category in which it fits. After the ANN is trained, it is capable of predicting in which category new triplets belong.

Such an oracle approach is appealing because of the ability to approximate functions. However, it has limitations to overcome. Input data may not be easily represented as characters and strings for use in ML. Also, deciding the structure of the network, such as the number of layers and neurons, may not be easy. The selection of training sets from test cases is another key problem that must be considered carefully: it should first be evaluated, which requires the use of other oracles.

3.3.4 Version-based Oracles

Version-based oracles are mechanisms that explore different version of the SUT to support decisions about the correctness of a test execution. The use of other versions of the SUT to implement test oracles was previously discussed in the introduction of pseudo-oracle concepts. This classification is very similar to the Derived Oracles, presented by Harman et al. [83]. Here, we briefly discuss different approaches to SUT versions as oracle information and how new versions can be used, namely:

- **N-version**: an N-version approach requires that N independently written versions of an SUT are also implemented [65]. If different outputs are produced by any version, a majority vote decides which output is likely correct. If the same team implements all versions of the SUT in the same language, it is possible that the same defect in the program will be present in other versions (correlated defects). To avoid such a drawback, Manolache e Kourie [139] suggest that different teams and language paradigms should be used. However, Knight e Leveson [116] found that even independently developed programs may contain correlated faults; and

- **M-mp (m-model Program testing)**: an M-mp approach is a variation of the N-version. In this case, only different versions of SUT are implemented, instead of a complete system, with the objective of reducing the test cost.

Other similar approaches of version-based oracles may include three variation of strategies. First, “regression testing” in which previously tested stable versions are used as oracle information to test programs developed by iterative processes [151]. Second, “third-party components” that can use tools such as extreme harvesting [94] to search for artifacts (as metadata) produced during normal development processes as the basis for component retrieval and matching on Web search engines. The retrieved components are then used as an oracle, and information. Third, one can mention oracles based on “mutation test” that uses outputs taken from programs generated from the original code with small modified
source code (mutant programs). Then, the oracle compares these outputs with outputs from the original code.

Section 4 presents a mapping with a quantitative analysis of studies on test oracles.

4 A Quantitative Analysis and a Mapping of Studies

Most test engineers agree that test oracles are a recent issue in software testing. Two decades ago, considering the huge number of conferences and journals on SE, the studies that dealt directly with test oracles were limited. They were confined to a portion of the literature addressing automated test strategies. Nowadays, this context is different and there are several researchers from academia and industry dedicating their efforts specifically to studies on test oracles. In this section, using a pool of more than 300 studies on test oracles, we map the oracle state-of-the-art in several ways:

- Counting the number of publications directly related to test oracles from 1978 to 2013;
- Mapping authors’ affiliations in order to measure the industry interest in test oracles and technology transfers from academia to industry;
- Presenting a list of academic efforts to test oracles (Ph.D. and Masters);
- Ranking countries of authors’ affiliations in order to discover the most active groups of researchers and universities;
- Presenting supporting tools;
- Analyzing the SUTs explored in empirical evaluation;
- Establishing a co-authorship network to understand the most prolific groups and highlight their interests; and
- Understanding the test oracle evolutions in terms quantitatively using line graphs of SUTs and approaches.

4.1 A Literature Review on Test Oracles

In the following section we present the methodology we have used to acquire a pool of 304 test oracle studies. We have created an online repository including all of these studies and important information such as title, authors, year and medium of publication, abstract, and others. After constructing the repository, we defined a series of data points to be collected from these studies in order to provide a bird’s-eye view of the issue. In this section we explain our methods and protocols for research and present the numbers that we derived from our analyses.

\[\text{Repository available in (draft): } \text{http://www.labes.icmc.usp.br/~rpaes/repo/repo.html}\]
4.1.1 Study Selection

Aiming to obtain a significant number of useful studies on test oracles, we searched for studies on indexed databases. In the context of this chapter, the term “study” means any written document published and available online such as, journal articles, conference papers, and technical reports (gray literature).

We have conducted web searches for studies in IEEE (Institute of Electrical and Electronics Engineers) and ACM (Association for Computing Machinery) using the following search string: "("test" OR "testing") AND ("oracle" OR "oracles")". We searched for these terms in three different fields: title, abstract, and keywords. We did not customize the searches for specific years, journals or conferences. All of this information is important for possible replications and future updates.

After making adjustments for each database, we collected more than 400 studies, combining ACM and IEEE results. After that, we read each paper’s abstract, analyzed the studies (excluding repeated entries) and selected the ones related directly to test oracles. In addition, this part of the analysis aims to identify off-topic studies such as research on Oracle® database systems. Studies on SE topics other than test oracles were excluded, as well. On the other hand, any study describing, examining, or developing any kind of test oracle was included in our analysis. We avoided imposing many restrictions on study selection in order to get a broad publication overview.

Following this manual selection we added some already known primary studies related to this topic. These papers did not appear in our original pool because they had appeared in publications not indexed by the two selected databases. Then, at end of the process described, we gathered a pool of 304 studies directly related to test oracles in an automated software testing scenario. One can access online the complete list of these studies and information about them through the repository web page. Figure 12 shows the work flow we have used in our selection process.

4.1.2 Study Classification

We have classified the selected studies according to the oracle taxonomy by oracle information characteristics, detailed in Section 3.3. This means that we have read each selected study, classifying them in four different categories: (i) specification based, (ii) metamorphic relation, (iii) machine-learning, and (iv) n-version or similar. During this analysis we found some secondary studies, which are studies that analyze other studies with new contributions and results (primary studies). Then, we classified these sorts...
of studies as (v) surveys. In our survey we identified secondary studies with classifications
or surveying test oracles aspects.

In order to classify all of the papers, we carried out a consensus process to decide on
pending papers, where the authors discussed whether such papers met our criteria. In
this context, a subset of studies was read by each author of this chapter. At first, based
solely on title and abstract, we decided about the study inclusion or not. Afterwards,
studies deemed as relevant according to goals were read.

4.2 A Quantitative Analysis on Studies

Analyzing our pool of studies, one can note that, in a generic way, the interest from
research of test oracles grew during the years. It is possible to note that the amount of
research about this field has increased year after year since 2008. Using a line graph,
Figure 13 presents a visual analysis of the number of studies included in our pool from
1978 to 2013. In addition, the same figure presents a table including the number of studies
identified each 3-year period from 1978 to 2013.

These quantitative aspects show a considerable evolution of research on test oracles.
The upshot of all this analysis is that the oracle issue is growing year after year, receiving
increasing attention from researchers. Research on test oracles has been quite constant
throughout the underlying 25-year period, since 1987. Considering the selectivity of the
conferences and journals and its broad coverage, it is fair to assume that research on test
oracles have been playing a substantial role in the SE context.

4.2.1 Study Analysis per Area

Our analysis and classification shows that research on test oracles have been published
under the five distinct categories in the following proportion: specification based ap-
proaches are responsible for 71% (217/304) of our selected studies; MR-based account for 12% (36/304); N-version and related studies account for 6% (17/304); machine-learning approaches represent 7% (20/304) of the approaches considered; finally, surveys compose 5% (14/304) of the studies. Figure 14 draws a fan plot that illustrates the differences among the approaches considered in this study. In addition, the same Figure includes a table that presents these numbers sorted from the highest to the lowest.

![Test oracle approaches](image)

Figure 14: Pie Chart of Test Oracle Approaches.

The line graph in Figure 15 illustrates the relative quantities and differences among studies selected for our study. At a glance, specification-based approaches have been in
a continuous evolution since 1995. Considering approaches related to MR, it is possible to note a considerable evolution after the year 2000. The evolution has established a regular constancy, showing that this area has been widely exploited by new research. Machine-learning approaches has maintained a certain constancy without many significant developments. Along those the same lines, the N-version and similar approaches appear as a supporting approach that did not show many evolutions. Finally, as a consequence of the popularity of automated testing activities, surveys on test oracles are becoming common among other approaches.

Figure 15: Year by Year Evolution of Oracle Approaches.

Complementing the analysis on the evolution of software testing approaches, Figure 16 shows graphically the number of oracle publications during each three year period since 1978. Regarding this visual information one can notice that since about 2004 the four approaches were firm and they appear together on subsequent research. Using this analysis, one can infer that sketches of research on N-version based oracles were noted first at the end of the 1980s and, after almost a 10-year period with no new results, these approaches came out again. Although it is true that after 2009 research has exploited more of these approaches. Finally, considering the visual information plot in this figure it is evident to note how much bigger the number of specification-based approaches when it is compared to other approaches.
4.2.2 SUT Analysis

Our analysis identified 87% (263/304) studies that have presented at least one experiment to support their findings about test oracles. Then, we designed an analysis to compare between practical and theoretical works (Figure 17a). We made a deeper analysis to ascertain methods and strategies used by researchers on test oracles to validate their approaches. In the context of this analysis, we considered all study cases, proof of concepts, examples of usage, quasi-experiments, controlled experiments and general empirical analysis as experiments.

The first experimental analysis preformed was directed to access the usage of “Toy Programs” and “Real programs”. Toy programs, which are very common in SE experiments, are systems with limited functions that generally are set to work for a specific purpose. Despite this fact, concepts applied to Toy programs may often be generalized for wide scenarios. In terms of this analysis, we considered the following as Toys: single specific functions, code excerpts, and SUT versions developed by the researchers. On the other hand, Real programs are commercial systems or open source systems available for final users.

Regarding the studies, 58% (152/263) used toys and 42% (111/263) exploited real software systems. Figure 17b presents a pie chart illustrating this comparison. In addition, Figure 17c presents the number of toy and real programs experimented in approaches published since 1990. An outcome observed was that results regarding experiments using...
(a) Theoretical versus Practical Studies. (b) Toy versus Real Programs.

(c) Toy and Real programs: year-by-year after 1990

Figure 17: Analysis on Validation Strategies and SUTs.

one or more real systems are much reliable. This is due to the fact that sometimes toys are generally of little practical use and less sophisticated than real programs.

Over 36 years of research, on new development systems on test oracles arose, as did APIs, programming paradigms, and languages with built-in tools. Regarding these changes, we accomplished an analysis to investigate test oracle research considering many systems implemented over these years. We have classified the SUTs in five distinct categories: (1) mobile/embedded or real-time reactive systems; (2) GUIs (Graphical User Interface); (3) general purpose; (4) Web and Internet applications (including SOA (Service-Oriented Architecture)); and (5) distributed and concurrent applications. Among general purpose we identified single functions and other popular SUTs and validation strategies such as, clock alarms, calculators, specific-domain tools, triangle classification, credit approval applications, ATMs (Automated Teller Machine) systems, cryptographic...
systems, numerical problems, and sort algorithms (bubble, quick, etc). Table 4 presents the number of SUTs per category identified in our analysis.

<table>
<thead>
<tr>
<th>SUT category</th>
<th>#</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) Mobile/embedded or real-time/reactive</td>
<td>44</td>
<td>17</td>
</tr>
<tr>
<td>(2) GUIs</td>
<td>15</td>
<td>6</td>
</tr>
<tr>
<td>(3) General purpose</td>
<td>164</td>
<td>62</td>
</tr>
<tr>
<td>(4) Web applications</td>
<td>35</td>
<td>13</td>
</tr>
<tr>
<td>(5) Distributed and concurrent</td>
<td>6</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 4: Number of SUTs and Categories.

About this SUT analysis, we highlight the number of Web application (13%, 35/264) and the number of embedded/reactive systems (17%, 44/264). Further, Figure 18 presents this analysis in a line graph that represents a distribution including the number of each SUT category by year from 1990 to 2013.

![Figure 18: SUT Categories Year-by-Year.](image)

4.2.3 Projects Analysis

Other aspects considered in our study were related to authors’ affiliation. We performed this analysis to raise two basic types of information:

- What is the participation of industry members and practitioners?
- Which countries have more universities and/or software companies researching test oracles?
Regarding this premise, we have checked all of the authors’ affiliations, classifying the studies as coming from academia (universities/colleges), industry (research institutes or private companies) or collaboration in cases of one or more authors from academia and industry in the same study. In this sense, we could note only 11% (32/304) of the test oracles studies selected belong to industry; 12% of the studies were classified as collaboration; 78% (237/304) of the selected studies were held in academic environments. Figure 19 presents this comparison using a fan plot.

Figure 19: Research on Test Oracles: Industry, Academia and Collaboration.

Regarding a temporal comparative analysis of studies from industry and academia, Figure 20 presents a graphical design of a year-by-year analysis between the numbers. This analysis shows a significant superiority of studies originated in academia. However, we noticed slight increase in industry and collaboration studies since 2008. This increase raises an important question: Are testing oracles becoming more applicable in industry scenarios? Possible answers to these questions are given in Section 5. To sum up this context we consider that the industry participation in research on test oracles is quite weak, however it is becoming as important as automated testing activities.

### 4.2.4 Demographical Analysis

In order to provide a wide demographic view about the research on test oracles, we have collected information about where each study published was conducted. We found the first author’s affiliated research institutes, universities/colleges, and private companies. Then, using the country of each professional address indicated on the publication, we counted the most active countries conducting research on test oracles. Table 5 presents the data collected from this analysis. We observed that together USA and China host about a half of the research on test oracles analyzed. USA has 104 (34%) and China has 42 (13%) of the studies considered in our survey. Considering the geographical analysis
Figure 20: Test Oracles: Industry, Academia and Collaboration.

provided, we highlight the research held in Australia, which represents 24 (8%) of the total studies.

Aiming to provide a wide view about each the contributions from each country, using a visualization tool [223], we designed a colorful world map demonstrated in Figure 21. In this world map the level of contribution of each country is presented using colors. Stronger colors represent countries with more contributions. Therefore, the color red represents the most prolific country and the color white represents no contribution at all. A brief visual evaluation of this map shows that India, Brazil, and Germany have contributed little so far.

4.2.5 Publication Strategies

This aim of the analysis is to verify the strategies used by researchers of test oracles and to communicate their new findings. Taking our repository of 304 studies, there were four Technical Reports and 300 studies published in conferences or journals of SE and related areas. Then, we took note of the means used of each publication in order to answer the following question: What are the most common venues and periodicals for research on test oracles?
Table 5: Most Prolific Countries by Authors’ Affiliation.

<table>
<thead>
<tr>
<th>Country</th>
<th>#</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>USA</td>
<td>104</td>
<td>34</td>
</tr>
<tr>
<td>China</td>
<td>42</td>
<td>13</td>
</tr>
<tr>
<td>Australia</td>
<td>24</td>
<td>8</td>
</tr>
<tr>
<td>Canada</td>
<td>19</td>
<td>6</td>
</tr>
<tr>
<td>France</td>
<td>19</td>
<td>6</td>
</tr>
<tr>
<td>UK</td>
<td>16</td>
<td>5</td>
</tr>
<tr>
<td>Germany</td>
<td>11</td>
<td>3</td>
</tr>
<tr>
<td>Italy</td>
<td>10</td>
<td>3</td>
</tr>
<tr>
<td>Malaysia</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>India</td>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>Brazil</td>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>Spain</td>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>Netherlands</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>Japan</td>
<td>3</td>
<td>0.9</td>
</tr>
<tr>
<td>Finland</td>
<td>3</td>
<td>0.9</td>
</tr>
<tr>
<td>South Korea</td>
<td>3</td>
<td>0.9</td>
</tr>
<tr>
<td>Austria</td>
<td>2</td>
<td>0.6</td>
</tr>
<tr>
<td>Taiwan</td>
<td>2</td>
<td>0.6</td>
</tr>
<tr>
<td>Switzerland</td>
<td>2</td>
<td>0.6</td>
</tr>
<tr>
<td>Others</td>
<td>11</td>
<td>3</td>
</tr>
</tbody>
</table>

Considering this investigation, before a more deep analysis, we can define that the number of conference studies (78%, 240/304) is four times higher than the number of journal studies (19%, 60/304). Figure 22 shows a fan plot that represents the number of periodical and conference publications.

Regarding the studies identified from periodicals, we can define that the most desirable target of test oracle research is the Journal *IEEE Transactions on Software Engineering*\(^7\) in this periodical we found more than 33% (20/60) of the studies, representing 20 studies. The second periodical most used by test oracle research is *Information and Software Technology*\(^8\) in which 10% (6/60) of the selected studies were published. Table 6 presents the complete relationship among studies and periodicals considered for this work.

Table 6: Test Oracle Studies Published in Periodicals.

<table>
<thead>
<tr>
<th>Journals</th>
<th>#</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEEE Transactions on Software Engineering</td>
<td>20</td>
<td>33.33</td>
</tr>
<tr>
<td>Information and Software Technology</td>
<td>6</td>
<td>10.00</td>
</tr>
<tr>
<td>ACM SIGSOFT Software Engineering Notes</td>
<td>3</td>
<td>5.00</td>
</tr>
<tr>
<td>Lecture Notes in Computer Science</td>
<td>3</td>
<td>5.00</td>
</tr>
<tr>
<td>IET Software</td>
<td>2</td>
<td>3.33</td>
</tr>
<tr>
<td>Software Testing, Verification and Reliability</td>
<td>2</td>
<td>3.33</td>
</tr>
<tr>
<td>IEEE Software</td>
<td>1</td>
<td>1.67</td>
</tr>
<tr>
<td>others</td>
<td>23</td>
<td>38.33</td>
</tr>
</tbody>
</table>

Considering studies published in conferences, we note a large variety of targets likely due to the existence of different conferences on software testing and automated testing. We highlight five main preferred targets followed by researchers on test oracles. Four of

\(^7\) see: [http://www.computer.org/portal/web/tse](http://www.computer.org/portal/web/tse)

\(^8\) see: [http://www.journals.elsevier.com/information-and-software-technology/](http://www.journals.elsevier.com/information-and-software-technology/)
these congresses are ICST (*International Conference on Software Testing, Verification and Validation*), ASE (*International Conference on Automated Software Engineering*), ICSE (*International Conference on Software Engineering*), and QSIC (*International Conference on Quality Software*), where were published 64 studies included in our analysis, 16 studies in each conference. Finally, the fifth most common conference found on our analysis was COMPSAC (*Annual International Computers, Software & Applications Conference*), where 15 studies were found. Table 7 draws a wide scenario about the most popular conferences among researchers on test oracles.

### 4.2.6 Prolific Researchers

One of the goals of this chapter is the identification of productive researchers on test oracles. Regarding the 304 studies, we identified 595 different authors. Only 149 (25%) of these authors have two or more studies considered in this survey. Taking this number as
Table 7: Test Oracle Studies published in Conferences and Workshops.

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Conference</th>
<th>#</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>ICST</td>
<td>International Conference on Software Testing, Verification and Validation</td>
<td>16</td>
<td>6.6</td>
</tr>
<tr>
<td>ASE</td>
<td>International Conference on Automated Software Engineering</td>
<td>16</td>
<td>6.6</td>
</tr>
<tr>
<td>ICSE</td>
<td>International Conference on Software Engineering</td>
<td>16</td>
<td>6.6</td>
</tr>
<tr>
<td>QSIC</td>
<td>International Conference on Quality Software</td>
<td>16</td>
<td>6.6</td>
</tr>
<tr>
<td>COMPSAC</td>
<td>Annual International Computers, Software &amp; Applications Conference</td>
<td>15</td>
<td>6.2</td>
</tr>
<tr>
<td>APSEC</td>
<td>Asia – Pacific Software Engineering Conference</td>
<td>12</td>
<td>5.0</td>
</tr>
<tr>
<td>ICSTW</td>
<td>International Conference on Software Testing Verification and Validation Workshops</td>
<td>10</td>
<td>4.1</td>
</tr>
<tr>
<td>AST</td>
<td>Workshop on Automation of Software Test</td>
<td>7</td>
<td>2.9</td>
</tr>
<tr>
<td>ISSRE</td>
<td>International Symposium on Software Reliability Engineering</td>
<td>6</td>
<td>2.5</td>
</tr>
<tr>
<td>ITNG</td>
<td>International Conference on Information Technology</td>
<td>4</td>
<td>1.6</td>
</tr>
<tr>
<td>TAIC PART</td>
<td>Testing: Academic and Industrial Conference – Practice and Research Techniques</td>
<td>4</td>
<td>1.6</td>
</tr>
<tr>
<td>SMC</td>
<td>International Conference on Systems, Man, and Cybernetics</td>
<td>4</td>
<td>1.6</td>
</tr>
<tr>
<td>SSIRI</td>
<td>Secure Software Integration and Reliability Improvement</td>
<td>3</td>
<td>1.2</td>
</tr>
<tr>
<td>ICSESS</td>
<td>International Conference on Software Engineering and Service Science</td>
<td>3</td>
<td>1.2</td>
</tr>
<tr>
<td>SEFM</td>
<td>International Conference on Software Engineering and Formal Methods</td>
<td>3</td>
<td>1.2</td>
</tr>
<tr>
<td>ISSTA</td>
<td>International Symposium on Software Testing and Analysis</td>
<td>2</td>
<td>0.8</td>
</tr>
<tr>
<td>SERE</td>
<td>International Conference on Software Security and Reliability</td>
<td>2</td>
<td>0.8</td>
</tr>
<tr>
<td>CISE</td>
<td>International Conference on Computational Intelligence and Software Engineering</td>
<td>2</td>
<td>0.8</td>
</tr>
<tr>
<td>Other events</td>
<td>101</td>
<td>41.6</td>
<td></td>
</tr>
</tbody>
</table>

a reference, we preview more contributions for the next years, once an expressive number of authors may be concluding more research on test oracles.

Among 595 different authors, according to our analysis, we highlight the most prominent and active:

- **Professor T. Y. Chen** – 14 studies (currently at Swinburne University of Technology – Australia), who has worked several years on metamorphic testing and some extensions in the areas of program testing, proving, and debugging;

- **Professor Paul Strooper** – 11 studies (currently at University of Queensland – Australia), who has worked on test oracles based on formal specifications;

- **Professor T. H. Tse** – 11 studies (currently at University of Hong Kong – China), who has worked mainly with test oracles considering metamorphic relation;

- **Professor Atif Memon** – nine studies (currently at University of Maryland – USA), who works with test oracle for GUI testing;

- **Professor ZhiQuan (George) Zhou** – seven studies (currently at University of Wollongong – Australia), whose studies are aimed at alleviating the oracle problem using, mainly, MR; and

- **Professor Wing-Kwong Chan** – seven studies (currently at City University of Hong Kong – China), who works with methodologies of pattern classification for metamorphic testing.
Regarding the authorship in a wider scenario, an analysis shows that only 6.2% (37/595) of the authors had four or more studies included. Table 8 presents the citation name and the number of studies of our survey for the 36 more prolific authors identified.

Table 8: Most Prolific Authors of the Survey.

<table>
<thead>
<tr>
<th>Author</th>
<th>#</th>
<th>Author</th>
<th>#</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bai, X.</td>
<td>4</td>
<td>Kim, P.-C.</td>
<td>6</td>
</tr>
<tr>
<td>Bieman, J.M.</td>
<td>4</td>
<td>Labiche, Y.</td>
<td>5</td>
</tr>
<tr>
<td>Briand, L.C.</td>
<td>5</td>
<td>Liu, H.</td>
<td>5</td>
</tr>
<tr>
<td>Carlington, D.</td>
<td>5</td>
<td>McDonald, J.</td>
<td>4</td>
</tr>
<tr>
<td>Chan, W.K.</td>
<td>7</td>
<td>Memmin, P.</td>
<td>6</td>
</tr>
<tr>
<td>Chen, T.Y.</td>
<td>14</td>
<td>Memon, A.</td>
<td>9</td>
</tr>
<tr>
<td>Chen, Y.</td>
<td>6</td>
<td>Murphy, C.</td>
<td>6</td>
</tr>
<tr>
<td>Guoerkei, R.</td>
<td>4</td>
<td>Peterson, D.K.</td>
<td>4</td>
</tr>
<tr>
<td>Harmon, M.</td>
<td>4</td>
<td>Schweigert, F.</td>
<td>5</td>
</tr>
<tr>
<td>Heimdahl, M.P.E.</td>
<td>4</td>
<td>Shahamiri, S.R.</td>
<td>5</td>
</tr>
<tr>
<td>Hoffman, D.</td>
<td>4</td>
<td>Staats, M.</td>
<td>4</td>
</tr>
<tr>
<td>Hoffman, D.M.</td>
<td>4</td>
<td>Strooper, P.</td>
<td>11</td>
</tr>
<tr>
<td>Huang, H.</td>
<td>4</td>
<td>Tonelli, P.</td>
<td>4</td>
</tr>
<tr>
<td>Ibrahim, S.</td>
<td>4</td>
<td>Tsai, W.-T.</td>
<td>6</td>
</tr>
<tr>
<td>Just, R.</td>
<td>5</td>
<td>Tse, T.H.</td>
<td>11</td>
</tr>
<tr>
<td>Kadir, W.M.N.W.</td>
<td>5</td>
<td>Xie, Q.</td>
<td>5</td>
</tr>
<tr>
<td>Kaiser, G.E.</td>
<td>5</td>
<td>Xie, T.</td>
<td>4</td>
</tr>
<tr>
<td>Kapfhammer, G.M.</td>
<td>4</td>
<td>Zhou, Z.Q.</td>
<td>7</td>
</tr>
</tbody>
</table>

4.3 Authors’ Collaboration

To measure the level of collaboration among the authors, we prepared a co-authorship network including all studies of our survey. This network matches all authors and co-authors who have published together at least one time. This network is graphically presented in Figure 23. Each node represents an author and each edge represents a collaboration between authors in at least one publication. It is important to highlight that in this network we do not consider unique authors. The network of authors and papers clearly identifies the collaboration level among authors. The distinguished members of each group are usually at the center of each clique. This visual organization shows the main research groups on the test oracle scenario, but it conceals quantitative information on the authors’ contribution.

A visual analysis of this co-authorship network reveals a huge group including more than 50 researchers (center of Figure 23). Visually, it is fair to assume that this is a result of the union of more than four major research groups. In addition, it is possible to note a huge group with more than 30 researchers (bottom of Figure 23). Besides these two groups, it is possible to note more than 15 small groups with at least five researchers. In a generic way, over the years, the trend is that these groups increase in number of researchers and join with larger groups. Thus, the exchange of experiences between groups anticipates excellence and improvement in future research.

In order to match the network presented in Figure 23 and the numbers presented by Table 8 we present a graphical association in Figure 24. Analyzing the figure, one can verify where the most prolific authors are located in a wide scenario. Each node represents
an author, and the sizes of the nodes are directly proportional to their number of studies. In addition, it is possible to note that a large amount of these authors are associated with the biggest groups identified. On the other hand, it is possible to note that there are six small groups (less than 12 authors) with an expressive number of contributions.

In addition, Figure 25 presents the co-authorship collaboration regarding each approach in our study classification (Section 4.1.2). Using this figure we reach a visual analysis about the maturity of collaboration for each test oracle approach.

The group of authors that have published contributions to the test oracles using MR (Figure 25a) seems to be very connected and collaborative. Note the connections between large groups of authors. Among the studies presented in this analysis, it is possible to note mainly two focuses: (1) test of programs without test oracles [16, 44, 45, 113, 124, 130, 162, 167, 168, 243], and (2) failure identification and classification [34, 35, 241].
The co-authorship network of studies on machine learning strategies (Figure 25b) differs totally from the metamorphic area for two main reasons. First, there is no connection among the research groups and the biggest group has only seven authors. Second, the groups identified use different approaches from different and huge areas such as Petri nets [127], “Backpropagation” method [228], Ontologies [187], artificial intelligence and other ANNs [5, 106, 108, 109, 133, 141, 200, 212, 250]. Then, it is possible to identify ten separate groups with at least four authors.

Only 17 studies related to N-version based test oracles (Figure 14). A co-authorship network could not be meaningful with this number of studies. Figure 25c represents the graphical collaboration among authors who played efforts to develop test oracles using N-version strategies. Among these strategies we highlight mainly: regression test [153, 188, 191, 224, 240], mutation analysis [182, 236], multiple version [17], N-version programming [203], and partial oracles [111, 112].

Due to the variety of approaches, it was expected that there would be more collaboration among researchers who have dedicated efforts on specification-based test oracles. One can confirm this through the co-authorship network of authors presented by Figure 25d. A visual analysis reveals three major groups of more than ten researchers and more than ten small groups with at least five authors each. Among the large number of approaches, we highlight: assertion based [47, 184, 185, 239, 258], bytecode-based and decision trees.

Figure 24: Active Researchers on Test Oracles and Their Respective Groups. (Generated using Cs2 [223])
(a) Metamorphic-Relation Based  
(b) N-version Based  
(c) Specification-Based  
(d) Machine-Learning Based  

Figure 25: Co-Authorship Network for Each Type of Test Oracle. (Generated using Cs2 [223])

... passive test oracles [206, 207], human oracle [148, 149, 209, 229], image based [57, 174], statistical analysis [18–20, 32, 254], log analysis [11, 227], ADA [80, 95], UML [29, 31, 126, 195], event-driven [150, 154, 155], natural languages [2, 208], and others [54, 158].

In addition to the approaches outlined above, many of the approaches presented explored formal specifications to alleviate the oracle problem. Among these formal specifications, we can identify a wide range of approaches. We highlight: finite state machine [14, 41, 84, 85, 181, 249, 253], algebraic specifications [13, 37, 136, 194, 259], object Z [135, 144, 146], model checking [39, 118, 119, 138], OCL [48, 176], Alloy [51], AspectJ [248], and Eiffel contracts [159].

4.4 Surveys and Position Studies

In our study we identified some works about complete or partial surveys on test oracles. In addition, we identify some opinion studies, also known as position studies, where the
authors’ point of view is expressed. Therefore, in this subsection we present the main points raised by these works.

Chronologically, one can consider the study presented by Weyuker [235] (already mentioned in Section 3) which was the first significant opinion paper about test oracles. In this study the authors present their considerations of the practical use of test oracles regarding cases in which an oracle does not exist or the tester must expend some extraordinary amount of time to determine whether or not the current output is correct. This kind of software system is called a “non-testable” program. In that study the authors highlight options of test strategies to be used instead of test oracles and their consequences.

In 1998, Hoffman [86] (already mentioned in Section 3) presented a study categorizing test oracles in classes regarding their various types of automated software verification and validation regarding real industrial approaches. In this analysis, the authors identified five different classes of oracles: True, Stochastic, Heuristic, Sampling, and Consistent oracles. In addition, in this classification the authors present comparisons among the advantages and disadvantages for each class covered. Later, the study Hoffman [87] was written by the same author with a different classification in which four types of oracle strategies (and not using any oracle) are identified and outlined: True, Consistency, Self Referential, and Heuristic.

In 2001, Baresi e Young [22] surveyed the proposed approaches related to test oracles that require neither pre-computed test cases nor previous versions of the SUT. Here the authors analyze test oracle strategies and approaches dealing with: Transducers, Embedded Assertion Languages, Extrinsic Interface Contracts, Pure Specification Languages, Trace Checking, and Log File Analysis. However, generally, this analysis is not comprehensive and it considers only a part of the research on test oracles.

In 2009, Shahamiri et al. [197] presented a comparative analysis among six categories of test oracles: N-Version Diverse Systems and M-Model Program Testing; Decision Table; IFN (Info Fuzzy Network) Regression Tester; AI (Artificial Intelligence) Planner Test Oracle; ANN-Based Test Oracle; and Input/output Analysis Based Automatic Expected Output Generator. The authors highlight that all of the methods mentioned have advantages and disadvantages and then they present an analysis aiming to compare the approaches regarding several aspects: Automation Tool, Automated Oracle Activities, and Limitations. The study concludes mainly two fundamental aspects to be considered about the oracle problem. First, it is not possible to completely automate the entire oracle process. Second, there is no unique approach to reach all oracle activities in different circumstances. Then, the upshot of all this study is that research on test oracles should develop a complete automated test oracle which is applicable in any type of software testing while all oracle activities are automated.
In 2011, Staats et al. [215] raised that several improvements to testing activities can be achieved regarding the problem of the oracle selection. Basically, this study proposes an analysis that includes test oracles in a revisitation of the fundamentals of testing in order to get better test results.

Chen et al. [40] revealed that oracle mismatches are one of the main causes of failed tests produced in automated testing. This study considered 197 failed tests produced in automated testing in GUI-based automated regression testing for a real industrial project in order to identify the main cause of false positives. Shende e Byagar [202] provided an opinion study in which automated test oracles are considered a fundamental challenge to be handled by testers in the software industry. In a similar approach, Itkonen et al. [99] presented a field study in which 12 testing sessions in four industrial organizations were recorded to measure how testers use knowledge while performing Exploratory Software Testing (ET). This study revealed that testers apply their knowledge of test oracles to determine whether a result was correct or not. The results showed that a considerable number of failures could be found outside the actual focus areas, showing the effectiveness of the exploratory testing approach.

Chan e Tse [36] summarized 15 years of their own research on test automation when a test oracle may not exist using MR. In order to do this analysis, the authors selected three relevant issues: (1) testing without a mechanism to determine the expected outcomes, (2) testing without a mechanism to gauge the actual results, and (3) testing without a mechanism to decide whether the actual results agree with the expected outcomes. Considering the first and the second issues, the authors presented their studies on metamorphic testing. About the third issue, the authors described their findings on pattern classification and formal object equivalence and nonequivalence.

In 2013, Harman et al. [83], trying to overcome the fragmented community of researchers and practitioners of automated test oracles, present a comprehensive investigation about the studies on a test oracle. The authors have constructed a repository of 611 publications on test oracles and their related areas. The survey covers specified, derived, and implicit oracles and techniques that cater to the absence of oracles. However, the main focus of this study is to provide an analysis of trends related to test oracles by means of a road map for future work at the interfaces between existing definitions. Among these trend analyses, the authors presented some future research direction like the potential of metamorphic testing.

4.5 Supporting Tools

During our analysis we found “tools” to support the oracle automation. We have considered as tools any means of automation associated with oracles, including specification...
language translators, development environments that support oracles, and frameworks. Table 9 presents a list of the most common tools we identified with a brief description and their references.

Table 9: Test Oracles: Supporting Tools.

<table>
<thead>
<tr>
<th>tool/framework</th>
<th>specification</th>
<th>references</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prosper</td>
<td>uses post-conditions</td>
<td>[28]</td>
</tr>
<tr>
<td>TROMLAB</td>
<td>formal specification for real-time systems</td>
<td>[27]</td>
</tr>
<tr>
<td>JUnit</td>
<td>unit testing for Java programs</td>
<td>[154, 155, 156, 157]</td>
</tr>
<tr>
<td>BIT</td>
<td>framework for automated black box testing</td>
<td>[24]</td>
</tr>
<tr>
<td>ISAO</td>
<td>uses machine learning to construct an oracle for image segmentations</td>
<td>[23]</td>
</tr>
<tr>
<td>ClassBench</td>
<td>a tool for executing and evaluating tests</td>
<td>[150, 151, 152]</td>
</tr>
<tr>
<td>DART</td>
<td>regression test for GUIs</td>
<td>[150, 153, 154]</td>
</tr>
<tr>
<td>Amsterdam</td>
<td>automates metamorphic testing</td>
<td>[152]</td>
</tr>
<tr>
<td>O-Flm</td>
<td>supports the test of programs with GUIs</td>
<td>[153]</td>
</tr>
<tr>
<td>JTOC</td>
<td>uses Java annotations and Java inner class to construct intelligent contracts for programmers</td>
<td>[154]</td>
</tr>
<tr>
<td>Gourlay’s framework</td>
<td>a theoretical framework for testing</td>
<td>[155]</td>
</tr>
<tr>
<td>Test templates</td>
<td>used to derive tests from model-based specifications</td>
<td>[156]</td>
</tr>
<tr>
<td>ASTRAR</td>
<td>group testing techniques for web services</td>
<td>[157]</td>
</tr>
<tr>
<td>TROMLAB</td>
<td>framework for rigorous development of real-time reactive systems</td>
<td>[27]</td>
</tr>
<tr>
<td>Circe</td>
<td>a grammar-based oracle for testing web applications</td>
<td>[20]</td>
</tr>
<tr>
<td>QUGIf</td>
<td>a Regression Testing Tool for Graphical User Interfaces</td>
<td>[49]</td>
</tr>
<tr>
<td>ZARRH</td>
<td>combines multiresolution static analysis and testing oracles</td>
<td>[50]</td>
</tr>
<tr>
<td>SPIN</td>
<td>a model checking tool</td>
<td>[51]</td>
</tr>
<tr>
<td>WST</td>
<td>generates test suites from Operational Specifications</td>
<td>[52]</td>
</tr>
<tr>
<td>EcoSuite</td>
<td>automatic test suite generation for java</td>
<td>[53]</td>
</tr>
<tr>
<td>ADEPT</td>
<td>executable specifications for automation behavior and user interaction</td>
<td>[74]</td>
</tr>
<tr>
<td>Host</td>
<td>a framework for table driven testing of java classes</td>
<td>[75]</td>
</tr>
<tr>
<td>TROT</td>
<td>automates the testing of equation execution</td>
<td>[76]</td>
</tr>
<tr>
<td>Zollay</td>
<td>a toolset for automatic fault localization</td>
<td>[77]</td>
</tr>
<tr>
<td>Fault Evaluator</td>
<td>experimental investigation of testing logical expressions in software</td>
<td>[78]</td>
</tr>
<tr>
<td>Warlock</td>
<td>generates oracles from Object-Z specification</td>
<td>[79]</td>
</tr>
<tr>
<td>Web Testing Explorer</td>
<td>automates the test of web applications</td>
<td>[80]</td>
</tr>
<tr>
<td>APTUSA</td>
<td>automates the invariant-based automatic testing</td>
<td>[81]</td>
</tr>
<tr>
<td>TGQ</td>
<td>oracles from specifications</td>
<td>[82]</td>
</tr>
<tr>
<td>F slices toolkit</td>
<td>a set of tools to test automata</td>
<td>[83]</td>
</tr>
<tr>
<td>tverage</td>
<td>conformance testing based on t-merge state machines</td>
<td>[84]</td>
</tr>
<tr>
<td>muJava</td>
<td>mutation test for Java programs</td>
<td>[85]</td>
</tr>
<tr>
<td>STS</td>
<td>statistical testing of software components</td>
<td>[86]</td>
</tr>
<tr>
<td>SafeRefactor</td>
<td>a technique to test Java refactoring engines</td>
<td>[87]</td>
</tr>
<tr>
<td>WebVisDr</td>
<td>automated oracles and analyzing test results of web applications</td>
<td>[88]</td>
</tr>
<tr>
<td>SpecBMM-ML</td>
<td>supports the generation of test cases from specifications</td>
<td>[89]</td>
</tr>
<tr>
<td>Mettloc</td>
<td>metamorphic testing from compilers</td>
<td>[90]</td>
</tr>
<tr>
<td>Ortra</td>
<td>automates unit-test suites with regression oracle checking</td>
<td>[91]</td>
</tr>
<tr>
<td>ConDiagnoser</td>
<td>an automated configuration error diagnosis tool for Java software</td>
<td>[92]</td>
</tr>
<tr>
<td>LETO</td>
<td>a feature-based oracle for airbus critical systems</td>
<td>[93]</td>
</tr>
<tr>
<td>PLASMA</td>
<td>a real-time verification based oracle for this system</td>
<td>[94]</td>
</tr>
<tr>
<td>Extreme Harvesting</td>
<td>a tool that collects components from internet</td>
<td>[95]</td>
</tr>
<tr>
<td>Cordesovy</td>
<td>converts metamorphic properties in testing methods that can be runned using assertions checking at run-time JML</td>
<td>[96]</td>
</tr>
<tr>
<td>PATHS</td>
<td>supports using UML and formal model on test oracles for GUIs</td>
<td>[97]</td>
</tr>
<tr>
<td>MD-TEST</td>
<td>verifies behaviors and internal data structure of GUIs</td>
<td>[98]</td>
</tr>
<tr>
<td>C#TeM</td>
<td>a runtime monitoring tool</td>
<td>[99]</td>
</tr>
<tr>
<td>CaStTest</td>
<td>test tool with support to castr specification based oracles</td>
<td>[100]</td>
</tr>
<tr>
<td>Dresden UCL Toolkit</td>
<td>interprets UCL constraints from a UML model and generates AspectJ code</td>
<td>[101]</td>
</tr>
<tr>
<td>NerveOutlet</td>
<td>causes different nice</td>
<td>[102]</td>
</tr>
</tbody>
</table>

4.6 Academic Efforts on Test Oracles (Ph.D. and Masters)

In order to provide a comprehensive analysis covering relevant academic efforts related to test oracles, we also searched for Masters theses and PhD dissertations that have made important contributions to the development of test oracles. These are listed in Table 10.

In Section 5, we discuss aspects related to the research on test oracles. This discussion is based on the analyses and numbers we have presented in the current section. We highlight some evidence and shifts revealed by our study. In addition, we raise some personal and particular points of view concerning the research on test oracles as a whole.
5 Discussions

The purpose of this section is to use the numbers eminent from Section 4 to introduce a discussion about possible areas for investigation by researchers. Briefly, the target of our discussion is an attempt to achieve advances in oracle approaches that fit test oracles in contemporary concepts, reducing human efforts on activities of verification. Typically, this discussion is related to our own point of view about present and future research on test oracles. Thereby, the content presented in this section express the opinion of the authors under their personal experiences and paradigms.

In this context, we raise three fundamental issues to be considered in contemporary concepts, reducing human efforts on activities of verification. Typically, this discussion is related to our own point of view about present and future research on test oracles. Thereby, the content presented in this section express the opinion of the authors under their personal experiences and paradigms.

In this context, we raise three fundamental issues to be considered in contemporary oracle approaches: (1) the high level of tool and specification languages; (2) the complexity of SUT’s outputs; and (3) the necessity of generalization of oracle strategies and properties. In the following subsections we present explanations and details to justify these issues. Next, we expand our discussion through the inclusion of some possible trends.

### Table 10: A Relation of Ph.D. and Master Work on Test Oracles.

<table>
<thead>
<tr>
<th>Year</th>
<th>Author</th>
<th>Title</th>
<th>University</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1993</td>
<td>Zhang</td>
<td>The construction of oracles for software testing</td>
<td>Durham University</td>
<td>Master</td>
</tr>
<tr>
<td>1994</td>
<td>Vercel</td>
<td>Enhancing debugging technology</td>
<td>Purdue University</td>
<td>Ph.D.</td>
</tr>
<tr>
<td>1995</td>
<td>Peters</td>
<td>Generating a test oracle from program documentation</td>
<td>McMaster University</td>
<td>Master</td>
</tr>
<tr>
<td>2000</td>
<td>Takahashi</td>
<td>An automated oracle for verifying gui objects</td>
<td>Florida Institute of Technology</td>
<td>Master</td>
</tr>
<tr>
<td>2000</td>
<td>Menache</td>
<td>Testing algorithmically complex software using model programs</td>
<td>University of Florida</td>
<td>Master</td>
</tr>
<tr>
<td>2000</td>
<td>Machado</td>
<td>Testing from structured algebraic specifications: the oracle problem</td>
<td>University of Edinburgh</td>
<td>Ph.D.</td>
</tr>
<tr>
<td>2001</td>
<td>Memen</td>
<td>A comprehensive framework for testing graphical user interfaces</td>
<td>University of Pittsburgh</td>
<td>Ph.D.</td>
</tr>
<tr>
<td>2004</td>
<td>Salmine</td>
<td>Scenarios specification as a testing oracle</td>
<td>West Virginia University</td>
<td>Master</td>
</tr>
<tr>
<td>2007</td>
<td>Feni</td>
<td>On the effectiveness of metamorphic testing for numerical programs</td>
<td>The University of Hong Kong</td>
<td>Master</td>
</tr>
<tr>
<td>2003</td>
<td>Moe</td>
<td>Observing the dynamic behaviour of large distributed systems to improve development and testing: an empirical study in software engineering</td>
<td>Linkoping universitet</td>
<td>Ph.D.</td>
</tr>
<tr>
<td>2004</td>
<td>Agarwal</td>
<td>A comparative study of artificial neural networks and fuzzy networks on their use in software testing</td>
<td>University of South Florida</td>
<td>Master</td>
</tr>
<tr>
<td>2005</td>
<td>Pacheco</td>
<td>Eclat: automatic generation and classification of test inputs</td>
<td>Massachusetts Institute of Technology</td>
<td>Master</td>
</tr>
<tr>
<td>2006</td>
<td>Edvardsson</td>
<td>Techniques for automatic generation of tests from programs and specifications</td>
<td>Linkoping universitet</td>
<td>Ph.D.</td>
</tr>
<tr>
<td>2006</td>
<td>Xie</td>
<td>Developing cost-effective model-based techniques for gui testing</td>
<td>University of Maryland</td>
<td>Ph.D.</td>
</tr>
<tr>
<td>2008</td>
<td>Yue</td>
<td>Is using images to test web pages the solution to a Sisyphean task?</td>
<td>University of Oslo</td>
<td>Master</td>
</tr>
<tr>
<td>2008</td>
<td>Doolhoe</td>
<td>Automated regression testing approach to expansion and refinement of speech recognition grammars</td>
<td>University of Central Florida</td>
<td>Master</td>
</tr>
<tr>
<td>2010</td>
<td>Park</td>
<td>An approach for oracle data selection criterion</td>
<td>The University of Minnesota</td>
<td>Ph.D.</td>
</tr>
<tr>
<td>2010</td>
<td>Lorano</td>
<td>Constraint programming for random testing of a trading system</td>
<td>KTH Royal Institute of Technology</td>
<td>Master</td>
</tr>
<tr>
<td>2010</td>
<td>Beasayah</td>
<td>A complementary approach for testing system correctness based on passive testing and fault injection techniques</td>
<td>University of Pierre and Marie Curie (Paris 6)</td>
<td>Ph.D.</td>
</tr>
<tr>
<td>2010</td>
<td>Jagannath</td>
<td>Reducing the costs of bounded-exhaustive testing</td>
<td>University of Illinois</td>
<td>Master</td>
</tr>
<tr>
<td>2010</td>
<td>Rose</td>
<td>Automated regression testing of ajax web applications</td>
<td>Faculty EEMCS, Delft University of Technology</td>
<td>Master</td>
</tr>
<tr>
<td>2010</td>
<td>Shrestha</td>
<td>An empirical examination of the effectiveness of phi properties as test oracles</td>
<td>University of Denver</td>
<td>Master</td>
</tr>
<tr>
<td>2010</td>
<td>Murphy</td>
<td>Metamorphic testing techniques to detect defects in applications without test oracles</td>
<td>Columbia University</td>
<td>Ph.D.</td>
</tr>
<tr>
<td>2011</td>
<td>Shahamiri</td>
<td>An automated framework for software test oracle based on multi-networks</td>
<td>Universiti Teknologi Malaysia</td>
<td>Ph.D.</td>
</tr>
<tr>
<td>2011</td>
<td>Zafar</td>
<td>Metamorphic testing research based on circular FSCS</td>
<td>East China Normal University</td>
<td>Master</td>
</tr>
<tr>
<td>2011</td>
<td>Kudari</td>
<td>An ontology-based automated test oracle comparator for testing web applications</td>
<td>Malmo University</td>
<td>Master</td>
</tr>
<tr>
<td>2011</td>
<td>Wu</td>
<td>Self-checked metamorphic testing of Monte Carlo simulation</td>
<td>East Carolina University</td>
<td>Master</td>
</tr>
<tr>
<td>2011</td>
<td>Liu</td>
<td>Metamorphic testing and its application on hardware fault-tolerance</td>
<td>University of Wisconsin</td>
<td>Master</td>
</tr>
<tr>
<td>2012</td>
<td>Asrafi</td>
<td>On testing effectiveness of metamorphic relations</td>
<td>Swinburne University of Technology</td>
<td>Master</td>
</tr>
<tr>
<td>2012</td>
<td>Jia</td>
<td>On Effective and Efficient Mutation Analysis for Cost and Integration Testing</td>
<td>University of Ulm</td>
<td>Ph.D.</td>
</tr>
<tr>
<td>2013</td>
<td>Nairli et al.</td>
<td>On test oracles for Simulink-like models</td>
<td>University of Sao Paulo</td>
<td>Ph.D.</td>
</tr>
<tr>
<td>2013</td>
<td>Afsah</td>
<td>A search-based generation of human readable test data and its impact on human oracle costs</td>
<td>The University of Shefyll</td>
<td>Ph.D.</td>
</tr>
<tr>
<td>2013</td>
<td>Vieira</td>
<td>A generic and automatic test strategy for compiler testing</td>
<td>Faculty EEMCS, Delft University of Technology</td>
<td>Master</td>
</tr>
</tbody>
</table>
5.1 High Level of Tools and Specification Languages

An objection in the research on the oracle problem of finding the middle-ground: what lies between the unfeasible complete automation and the manual means? This is a question that is hard to answer. There are many variables that can influence such a trade-off. Therefore, we believe there are many paths to be explored. Moreover: how to define a solution of (partially) automated oracle that is welcome in the software industry?

Based on our analysis, generically, a testing team may be reluctant to adopt a solution that is too hard to learn or too intrinsically domain/technology dependent if such variables change frequently. Even for long term projects and test benches, SE methods teach us that it is sometimes difficult to change mature patterns when they have been applied for a long time. Tools regarding test oracle approaches seem to be complex. A tool should be attractive to its user, that is, easy enough to be used and extensible to allow new technologies to be added in the future. Due to this, it is possible to note that the acceptance of research on the oracle problem by the industry is a challenge. This problem of technology transfer became clear from the numbers provided by the comparative analysis among studies from industry, academia or collaborations (Figure 19).

We believe that a possible solution to the problem of technology transfer is the development of effective methodologies to transfer results of test oracle research to industry. It seems that a considerable amount of researchers on test oracles do not produce products from their findings or thesis. Many times impressive and impactful studies on the oracle problem are published but their practical exploration are not well presented and documented. Disregarding some exceptions, this lack of products is due to the fragmentation of the research groups. We feel that, despite some collaboration among research groups (Figure 23), there is a lack of cooperation to create a product with several resources.

Another point to be considered in dealing with problems related to technology transfer is the supply of training materials and tutorials about practical approaches. Available frameworks, tools, and APIs seem to be difficult to handle for testers who are not involved in their original design. Tutorials and practical examples are not usually mentioned by the authors of studies. Likewise, internet repositories including code samples and templates are not common. Due to this, practitioners and researchers interested in following new strategies have to expend a lot of time to learn to adopt new concepts and paradigms.

In addition, in our analysis, we observed many studies on parts of an oracle, but they do not treat the problem as a whole. For example, there are many studies on specification languages that discourse on how expressive they are with regard to other languages, their applications in different domains or how it is possible to translate them to automata and how to turn expressions more efficiently. But there is no study on the tracing from a
document to a specification language that is used as oracle information and how expressive a specification language actually should be to fit as oracle information. In addition, there is a lack of studies on how to facilitate the specification writing for complex requirements in the oracle context. A similar situation occurs for MR where describing metamorphic testing can be a manually intensive technique for complex cases [167].

All the timely challenges, namely, the difficulty in generalizing automated solutions, in learning a specification language and writing requirements, usability and support mechanisms, initial configurations, input and output parameters, hamper the oracle automation and makes its acceptance difficult in the industry. Such points must be considered in a context where an oracle procedure must analyze its information to allow the automation. For instance, on the machine-learning research, there are many studies that demonstrate their role as oracles, but there is not much effort on presenting how to prepare the data of an SUT to feed the network.

5.2 Complexity of SUT’s Outputs

Another delicate issue to be faced by researchers and practitioners on test oracles is the complex format of the SUT’s outputs. Complex formats are those for which the decision on their correctness requires sensory and perceptual aspects of a human being, such as vision and human hearing. For example, systems with sophisticated GUIs, some web applications and virtual reality environments have typical instances of these complex outputs. Moreover, our analysis has revealed few practical studies directed to these systems (Figure 18 and Table 4).

Systems with complex outputs are becoming common nowadays. The competitiveness makes the software industry develop ever more attractive and exquisite outputs. Then, naturally, the complexity of outputs is inherent from this generation of contemporary software. In addition, the contemporary software has a vast array of technologies including the hardware platforms, operating systems, and programming languages to support complex scenarios.

Testing technologies can be updated in the same proportion to this “new generation of systems” bringing benefits to final users. In particular, test oracles are supposed to support a productive evaluation of complex outputs. To illustrate this situation, when an SUT has a value or string as an output, the oracle rule is alleviated. Then, testers are able to define automated approaches in a “friendly” way. On the other hand, when the system output is given in a complex format, the oracle problem involves an “extra problem” – defining adequate methods for productive automation.

Following, we present several output formats that could make the oracle problem worse:
• Web applications: they should work well visually in several browsers. For final users it is common to see, “text fields”, “checkboxes” and “comboboxes” not working properly under different browsers. Visual non-detected errors could be defining the failure to a company;

• Virtual Reality (VR) or Augmented Reality (AR) systems: these systems could carry visual errors, regarding positioning and dimensioning of virtual objects in the three dimensional environments. Automated testing strategies for these applications are highly manual and ad hoc;

• Text-to-Speech (TTS) systems: a system whose outputs are provided by means of an audio signal (such as, representing speech) are commonly evaluated by human audition and scoring. Then, the automation of test oracles for this context can be considered an abstract and hard task. Despite the complexity related to the test, TTS systems have several applications, such as support for computer-human interaction for blind people, automatic reading of e-mails in embedded and mobile applications, reading of social network updates, and call center automation;

• GUIs: software systems whose output is promoted by means of a GUI (including mobile applications) must work properly independently of platform, screen resolutions, screen orientation, touch screens sensibilities, color systems, monitor settings, and Look and Feels (L&Fs). In addition, event-driven systems are a well know problem when the issue is processes of quality assurance. In this way, the complexity associated with SUT’s outputs is transferred for automated test strategies. For instance, generally, rippers, and record/playback tools are necessities;

• Images or graphical outputs: systems whose outputs are given by a processed image, such as, CAD (Computer-Aided Diagnosis) systems are naturally hard to be tested automatically. This kind of system, generally, is designed to support decisions in diagnosis. Visual analysis is necessary for CAD system testing; and

• Videos: systems whose outputs are given in video formats require synchronization between sounds and frames of images. Quality aspects have to be verified independently of platforms. Additionally, these systems may need to be associated with subtitles for videos and delays can occur. As a result, these complex characteristics limit the alternatives for test automation and, consequently, test oracles.

Implementing automated test oracles for SUT with complex outputs requires the application of specific techniques. These techniques might be directed at evaluating specific characteristics of the SUT’s outputs. Further, these characteristics should represent the
main essence of the output as whole, working as information sources for testing activities. For example, regarding complex GUIs or images, these characteristics may be represented by colors, contrast, forms, textures, or components’ positioning. On the other hand, considering TTS systems, these characteristics may be represented by naturalness, volume, energy, duration, pauses, etc.

In this context, an expressive contribution can be held by means of the composition and update of repositories including open source tools, frameworks and APIs on the test oracle. Regarding our pool of more than 300 studies, few researchers provided their resources and technologies to access the internet. Further, when the researchers provided their products for public access, generally, there were not enough support materials for other researchers using/extending these tools and frameworks. However, we believe tool repositories would help to advance the state-of-the-art faster. Since open source tools could be improved and extended, this would help avoiding groups starting new research from scratch.

5.3 Generalization of Test Oracle Strategies and Properties

It is necessary to make a generalization of test oracle research. This necessity was introduced by Harman et al. [83]. We believe this generalization should be held under two fundamental aspects: (1) theoretical, by means of oracle properties and (2) practical, by means of generic oracle strategies. Below, we present aspects associated with these two generalization perspectives:

- Theoretical generalizations: future studies on test oracles may consider the standardization of theoretical definitions and properties. Section 3 presented plenty of definitions and categories of test oracles. Future research may be focused on establishing more general theoretical definitions and properties on test oracles. We see several benefits associated with this standardization:
  - More broad definitions and taxonomies for test oracles: generally, each test oracle researcher presents particular definitions closer to their own interests. This makes the research on test oracles a fragmented issue, instead of continuous;
  - Possibilities of Systematic Literature Reviews (SLRs): SLRs are a means of aggregating knowledge about an SE topic or research question. In other words, SLR is a means of identifying, evaluating, and interpreting all available research relevant to a particular research question, or topic area. It is useful for researchers and practitioners to find out techniques and strategies in their particular interest area. In SE, generally, the study selection for SLRs is performed by means of a research string with a set of key words in an indexed basis
of scientific articles [64]. Then, for specific domains and contexts, theoretical generalizations of test oracles may facilitate the access of the state-of-the-art; and

- Maturity: well explored areas of software testing have standards and patterns for their basic properties and definitions, such as mutation testing, model testing, and regression testing. Standardized definitions of properties are propitious to research collaboration and group integrations.

• Practical generalizations: as we presented in Section 2 of this chapter, test oracles are particular processes useful in a specific testing scenario. Due to these processes, new research on test oracles may include strategies that offer support to more than one field output test. We present two consequences associated with the generalization of test oracle strategies:

  - Flexibility: due to the variety of domains, researchers should consider the development of more adaptable tools and flexible frameworks. A possible way to realize the tools and frameworks is the implementation of plug-in-oriented resources and technologies. In this context, a plug-in represents a software component that adds a specific testing feature to an existing tool/framework; and

  - Integration of tools and frameworks: considering the large number of tools and frameworks presented in Table 9, research to generalize approaches to oracles would be favorable for the possibility of integration in frameworks and tools.

Using our survey as background, in the following section we present particular thoughts about future research on test oracles.

5.4 Trends on Test Oracle

The investigation of trends presented in this section is based on the quantitative analysis obtained through the evaluation of the evolution of oracle approaches (Figure 15) and the type of SUT exploited in experiments (Figure 18). Next we present some trends in a high level description regardless of technical aspects associated with each trend.

The first trend to be considered by researchers on test oracles is the conduction of “multidisciplinary approaches”. Current SE projects associate different forms and concepts from distinct areas of computing. For instance, the association of ANN and software testing can result in significant evolutions, alleviating human oracle costs and to automate test oracles. These evolutions are results of a well conducted multidisciplinary project
associating concepts of AI and SE. Regarding test oracles, one can mention future research associating concepts of Processing Image (PI) for checking results of SUTs which include GUIs, web application and images in their outputs. In the same context, techniques of Signal Processing (SP) can be included in testing strategies of systems of voice synthesis or TTS systems.

A visible trend on test oracles is the “improvement and development of new strategies associated to metamorphic testing”. Besides a connected group of researchers (Figure 25a), our analysis demonstrated a considerable evolution of research on metamorphic testing over the last five years (Figure 15). Research efforts in metamorphic testing have branched out into two main directions: (1) applying metamorphic testing to programs in different domains as a solution to overcome the oracle problem and (2) improving/extendng the metamorphic testing technique itself, such as automatic detection of MRs and utilizing MRs in fault localization. In this sense, through proper associations with traditional concepts of the test, metamorphic testing must fit itself as a consolidated solution to alleviate the oracle problem.

Another trend is the “integration of the test oracle in rippers and testing tools developed for ubiquitous and pervasive systems”. Ubiquitous and pervasive systems compose an expressive advance in computing that involves the development of applications to be executed everywhere and anywhere. In practical terms, to support these advances, one can consider the different embedded systems and mobile devices available nowadays. Appropriate automated test environments (including a test oracle) for ubiquitous and pervasive systems have to ensure testers that the application is able to run properly under different conditions, such as operational system versions, screen resolutions, screen orientations, and hardware configurations. Technically, this trend might consider the development of integrated environments with SUT and testing technologies. These environments might allow several dynamic changes and adaptations in both SUT and testing technologies.

Finally, using the analysis presented in Section 4, our study has revealed a trend of test oracle research towards the “exploration of secondary studies”, such as surveys and judicious literature reviews. Secondary studies are useful tools to support decisions by researchers and practitioners about the correct technologies to be applied in their projects. Over more than 30 years of research on test oracles, there are several approaches available in the literature. Then, before defining their testing strategies, professionals may use secondary studies to support their decision about which test oracle they might use. According to Figure 15, this trend can already be visualized as a result of the last three years of research.
6 Final and Concluding Remarks

Test oracles perform an elementary function on testing activities – to ascertain whether a test execution is correct or not. In automated testing scenarios, test oracles may be implemented to ensure the productivity to check the correctness of SUT’s outputs. Otherwise, the process of verifying test outputs should be conducted by testers, requiring several human efforts. Once a testing designer has a well implemented test oracle, it is possible to design huge test sequences including more test cases in a systematic approach. Hence, these increase the reliability that the SUT is close to its specifications. However, it is not always a simple task to estimate and evaluate test outputs. In some cases, due to the complexity of the SUT requirements or output, even a human oracle expends a lot of time defining whether test results are acceptable. Given the relevance of this issue, several groups are trying to develop practical approaches for incorporating test oracle mechanisms in their testing processes. Due to this, over more than 30 years of research on test oracles, one can identify clear shifts in the research literature. We have therefore surveyed more than 300 studies on test oracles to measure and report these shifts and predict some trends for future evaluation.

In this chapter, besides some trends and discussions, we have presented a quantitative analysis of a huge pool of studies on test oracles. We have categorized the approaches reported and their experiments. In addition, we present an amount of data sufficient to measure the state-of-the-art on test oracles. To support our study, we have graphically presented: approach evolution, SUT classification, identification of research groups, co-authorship of networks, collaboration maps, a broad taxonomy commonly found in the literature, a critical view about issues to be regarded by contemporary oracle approaches and other analyses. Further, we shared out particular thoughts about trends to be faced by researchers and practitioners test oracles in the next years regarding contemporary software systems. In this sense, under a general and high level description, our goal was to offer the knowledge gathered by the authors through the years of research on that field and point to what may be the next challenge: defining solutions which will shape the coming state-of-the-practice.

Our final analysis, giving us an upshot of all this, states that test oracles are obtaining more and more space in industry projects. Daily, researchers have been dedicating their efforts to set a space for test oracles among standardized testing concepts. Future software testing activities will need a verdict about the output of the SUT – requiring a test oracle mechanism. This provision is clear and primary, then, test oracles are welcome in contemporary test projects. Due to this, we strongly encourage everyone who is starting studies on test oracles to read this chapter. Then, we believe that quantitative analysis and taxonomy specifications are a good reference to be extended for students and practitioners.
In addition, through the reading of this chapter, researchers and practitioners can notice that there is a wide field of research yet to be explored. As for the integration of already known and well-established approaches aiming at the industry acceptance, this fact enriches the monograph presented by means of this chapter.

Acknowledgements

The authors are grateful to Marcio E. Delamaro and Fatima L.S. Nunes for valuable comments. A thank you to Norine Walker, Juliana Oliveira and the UMD writing fellows for their accurate reviews and comments. Currently Rafael is a student visitor at University of Maryland and Prof. Atif Memon is his supervisor. Rafael is supported by FAPESP (Fundação de Amparo à Pesquisa do Estado de São Paulo) – Grant Numbers 2012/06474-1 and 2013/01775-6.

About the authors

Rafael A. P. Oliveira is a Ph.D. candidate at University of São Paulo (USP/ICMC). He has received his M.Sc. in 2012 from the University of São Paulo. He is a visiting scholar at the University of Maryland (UMD, College Park, MD, USA), where much of this book chapter was conducted. His research focuses on alleviating human testing efforts, automated and systematic testing strategies, automated test oracles, GUI and TTS testing. In addition, his main interests are software testing, quality, and reliability. His main publications can be found in his personal site (http://www.labes.icmc.usp.br/~rpaes/) and on Lattes Platform (http://lattes.cnpq.br/0793753941171478).

Upulee Kanewala is a Ph.D candidate in the Computer Science Department at the Colorado State University. Her research interests include developing automated test oracles, effects of test oracles in testing and automatic detection of metamorphic relations that can be used as test oracles.

Paulo Augusto Nardi is Ph.D. in Computer Science and Computational Mathematics at the University of São Paulo, Brazil. As part of his doctoral studies, from 2011 to 2012, he was visiting scholar at the Politecnico di Milano, Milan, Italy. He received his M.S. in Computer Science from UNIVEM in 2006. His current research interests include software testing, test oracles, empirical software engineering, and embedded systems. His main publications can be found on Lattes Platform (http://lattes.cnpq.br/5154586471030104).
References


In: Proceedings of the 4th IEEE International Conference on Software Testing, 
Verification and Validation (ICST 2011), Berlin, Germany, 2011, p. 110–119.

[206] Shukla, R.; Carrington, D.; Strooper, P. A passive test oracle using a component’s 
API. In: Proceedings of the 12th Asia-Pacific Software Engineering Conference 

[207] Shukla, R.; Strooper, P.; Carrington, D. Tool support for statistical testing of 

[208] Smith, B.; Williams, L. On the effective use of security test patterns. In: 
Proceedings of the 6th IEEE International Conference on Software Security and 

2nd Workshop on Software Testing, Verification, and Analysis (STVA 1988), Banff, 

[210] Soares, G.; Gheyi, R.; Massoni, T. Automated behavioral testing of refactoring 

tool for applying automated oracles and analyzing test results of Web applications. 
In: Proceedings of the Testing: Academic & Industrial Conference – Practice and 

[212] Sprenkle, S.; Hill, E.; Pollock, L. Learning effective oracle comparator combinations 
for Web applications. In: Proceedings of the 7th International Conference on 

of the 21st Digital Avionics Systems Conference (DASC 2002), Irvine, CA, USA, 

[214] Staats, M.; Gay, G.; Heimdahl, M. Automated oracle creation support, or: How 
I learned to stop worrying about fault propagation and love mutation testing. In: 
Proceedings of the 34th International Conference on Software Engineering (ICSE 


Applications Conference Workshops (COMPSACW 2011), Munich, Germany, 2011, p. 137–142.


