A Survey on Test Oracles

Paulo A. Nardi and Eduardo F. Damasceno, *DACOM, Universidade Tecnológica Federal do Paraná - UTFPR*

***Abstract*—Oracles play a key role in software testing since they state the correctness of the software under test. Unfortunately, many testers tend to judge their own executions, but human oracles are error-prone, slow, and very expensive. In contrast, automated oracles are usually faster, cheaper, and much more reliable: many proposals have addressed the problem, but no unique solution has emerged so far. The variety of these solutions motivated the survey of the state of the art on test oracles presented in this paper. Besides introducing the general concepts, we propose a classification based on the origin of the information used by the oracle. For each oracle type, the survey discusses it, proposes some examples, and highlights its limitations. Moreover, it compares the different types and provides a final holistic assessment. The goal is to provide software testing professionals and researchers with a taxonomy, a critical overview, and a fair assessment of the state of the art in the domain.**

***Index Terms*—Test Oracle, Software Testing, Test Automation.**

I. Introduction

Test oracles occupy a prominent role in the testing process: they determine whether the software under test behaviors correctly [1]. Oftentimes, testers themselves assess the correctness, but they are usually slow, error-prone, and very expensive. In contrast, automated means are cheap and can give better, faster and more reliable results.

Sharamiri [2] states that a "complete" test oracle is in charge of generating the expected outputs, running the test cases, comparing expected and actual outputs, and deciding whether the execution is correct. While running the test cases is simple, the other three activities can be carried out in a variety of ways and effectiveness. The identification of expected results is often not trivial, and many test data may not produce any relevant output. To name drawbacks on the comparison activity between expected and actual results: programs may calculate outputs that are unknown (for example, the identification of as many decimal numbers as possible of π); the expected output may be too large or complex to be compared in reasonable time by manual means and too difficult to be specified and automated.

The difficulty in deciding whether a result is acceptable or not is known as the Oracle Problem [3]. Weyuker [4] was the first to state that programs without pre-computed results are non-testable. Nonetheless, modern test oracles aim to identify expected results, assess obtained outcomes, and decide whether the execution is correct for both testable and non-testable software. Despite they are not "ideal" oracles in the sense they cannot guarantee the identification of all possible errors, different approaches can improve the test activity if compared with manual means.

The many facets of the oracle identification problem motivated us to conduct a systematic review on the approaches proposed so far to identify the different types of oracles along with their limitations. We classified the oracles based on the origin of the information used to decide whether the program under test is correct or not: (i) specification-based oracles; (ii) metamorphic relation based oracles; (iii) machine learning based oracles; and, (iv) N-version oracles.

The rest of this paper is organized as follows. Section II introduces the basic concepts about test oracles. Section III presents a summary of how the systematic review was executed. Section IV presents the actual overview where we classify and describe the approaches found in literature. Section V provides a discussion on the oracle classes. Section VI concludes the paper.

II. General concepts

This section presents basic concepts to the understanding of this paper. Given that test oracles are classified in many ways, it is also presented common types of oracles found in the literature.

There are programs which produce large number of outputs, and thus checking their correctness becomes time consuming and error prone. This is why we need methods and techniques to produce automated oracles efficiently.

An ideal automated oracle should be able to mimic exactly the behavior of the application under test in a completely reliable way: it should accept all possible inputs and produce its respective results, correctly [5]. A practical test oracle, however, does not need to target all inputs and outputs, but it could concentrate on the pairs used on the tests.

A test oracle should also have knowledge of the expected output and compare it with the actual outcome (also named here as obtained output). This means that an oracle should comprise both information and procedures [6]. Oracle information represents the expected output, which can be defined through specifications, acquired by previously stored results, from execution of code or models developed in parallel, metamorphic relations, or machine learning. Information can provide concrete results (values), that is, the exact value of a result; or constraints that must be respected, otherwise a result would be considered wrong or unacceptable. An oracle procedure compares the oracle information with the

obtained output: this comparison can be performed either at runtime or off line, that is, after the execution [7].

Different oracle information implies different oracle procedures and may also influence the effectiveness of the oracle. Specification-based oracles use system specifications to compare whether a program is correct; metamorphic relation based oracles apply known relations between the inputs and their respective outputs, which are not necessarily part of the system specification; oracles based on Machine Learning attempt to mimic the program's behavior - as a function approximator with previous test cases; N-version oracles use different implementations of the software under test as oracle and, in the presence of divergent results, the expected results is the one with the highest number of occurrences.

Roughly, we can also classify the different proposals in two big families: pseudo-oracles and partial oracles.

Pseudo-oracles are programs (executable models or code) written in parallel with the code under test [8]. The goal is to run the oracle and the software under test with the same input data and compare the obtained outputs. The program passes the test when results are the same on both, or if they are within an acceptable margin of accuracy. Besides the burden of a double development, there is no guarantee that the oracle is correct and does not suffer the same problems of the program under test.

Partial oracles are able to identify if a result is incorrect, even without knowledge of the correct output [4]. The verification is based on specifications, written as constraints such as contracts (pre and post-conditions) and invariants [9]. Pre-conditions, post-conditions and invariants are expressions that must be satisfied, respectively, before, during and after executing the program under test. For example, a partial oracle for a program that calculates the sine function can be based on the following post-condition: the result should be within the interval [-1:1]. Any result outside this range is reported as an error.

Oracles can be also classified as active or passive. An active oracle mimics the behavior of a software under test [10], e.g., neural networks or executable models. Passive oracles check the behavior of a component, but they do not reproduce its behavior [11]: e.g., specification-based or metamorphic relation based oracles.

III. Systematic review

We conducted a systematic review on different approaches, proposals, and solutions available for test oracle generation and use. It aims to classifying the types of automated test oracles, their limitations and provide guidelines to researchers and testers (a list of support tools were listed by Nardi [12]). An update on the systematic review was provided for further work [13]. We searched five digital libraries: IEEE, ACM-Digital Library, Spriger, Scirus and Scopus. The selected papers had at least one of the following characteristics: describing how a test oracle can be generated automatically; identifying a test oracle and its definition or application; identifying tools that support oracles; addressing the limitations of oracle utilization.

We obtained 493 papers, but some were copies. Thus, we eliminated duplicates, results that had no significant keywords in the abstract or title, unavailable papers and those not written in English. After the pre-selection, we read all papers, discarded the ones which not follow the search criteria, and kept 304 papers: 217 papers focus on specification-based oracles, 36 mention oracles based on metamorphic relations, 17 papers mention n-version or similar oracle techniques, and 20 papers focus on machine learning based oracles.

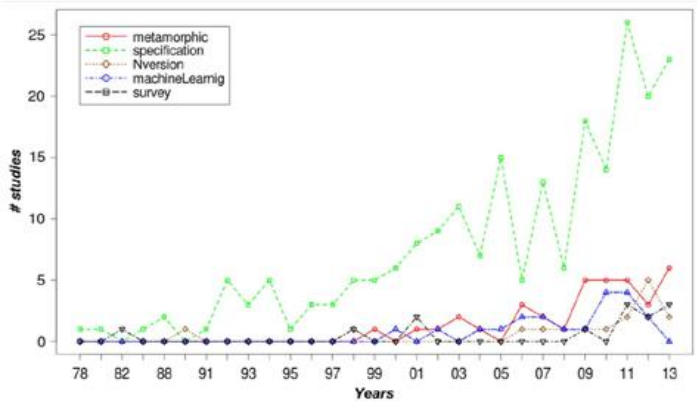

Fig. 1. Publications by year

Figure 1 presents the number of publications by year. We observed a heightened interest on test oracles in the last 10 years, notably after 2001.

IV. A classification by oracle information

As previously described, different oracle information implies different oracle procedures and may also influence the effectiveness of the oracle. This section describes the identified categories.

A. Specification-based oracles

The formal specification of a system provides a source of information about the correct behavior of its implementation and thus it is a valuable source for test oracles [14]. There is a wide range of different specification approaches and languages: for example, models, assertions, extrinsic interface contracts, and pure specification languages; Baresi and Young [15] have already surveyed some significant proposals. These oracles have also been applied on “real-world” applications for example by Volvo [16], the U.S. Department of Defense [17], and Microsoft [18].

If a specification is executable - e.g., Simulink models or state machines-, the tester can use it as oracle and compare the outputs obtained from the software with those produced by the specification. For example, Andrews [19] and Tu [20] use a specification language called LFAL (Log File Analysis Language) to describe state machines. A state machine description file (SMD) represents the specification of the software under test. A parser uses the SMD to generate the analyzer. The execution of the software under test creates a log

of events: the analyzer highlights a failure every time there is a mismatch with what is expected by the state machine.

Some tools, as Simulink, can generate code automatically from a model, which removes the need for manual coding. A convenient feature of such tool is that a model can be executed and its behavior can be analyzed prior to the code generation. The complexity of such model, however, imposes their verification: oftentimes one uses higher level specifications as a Simulink model, and the same issue remains: how can one compare the outputs produced by a model or by the code with its specification? A possible solution is the use of partial oracles, that is, constraints are used to analyze whether an output generated by the model execution is acceptable. This solution may be seen as a simplified way to check the model by focusing on its critical aspects. These constraints should be described formally to eliminate any ambiguity.

Examples of specification languages that are used to describe such constraints (or rules) are: Z, Object Z, OCL, Eiffel, VDM, JML, state machines, SDL and MITL.

The rest of this section discusses the trade-off between using parts or complete specifications as oracle information, the different approaches for comparing the outputs produced by the implementation and the oracle, and some examples of oracles in particular domains.

System and oracle specifications: since in many cases, a tester only needs to check automatically a subset of a system specification (usually the most critical parts) the separation between system specification and oracle specification can be useful and appropriate. The whole system can be specified at a higher abstraction level and in a rather informal way, while the parts of interest can be transformed into a very detailed and precise specification, which can be used as oracle information. An oracle procedure, then, must compare the obtained result against the formal specification. Therefore, such procedure must be able to interpret the formal specification.

Mapping oracle information to implementations: to check the implementation against a formal specification (oracle information), one needs to transform concrete values; those that come from the implementation, into abstract ones, which are in the domain of the formal specification. To this end, a retrieve function is in charge of mapping concrete input data to abstract inputs, and concrete output data to abstract ones.

There are several approaches to such mapping. Some specification languages, such as Z, Object Z, and algebraic specifications allow the representation of classes. These classes are referred as abstract data types [21].

According to Peters [22], there are restrictions on writing a specification that can be used as oracle. Programming languages are usually richer than specification languages and thus the latter must often mimic some of the constructs that are freely available in the former. For example, the use of primitive relational operators like "=" is valid only for basic data types; more complex types, such as structures and objects, require the operator be defined through auxiliary predicates.

Similarly, many formal languages do not provide the concept of null object, which is quite a standard feature of many programming languages. These restrictions may hamper the mapping between specification and implementation.

To bypass some of the mapping complexity, one could use assertions in the form of pre- and post-conditions and invariants, that embed the formal specification: rigorous expressions embedded in the implementation by specifying the circumstances and conditions under which the execution can be considered correct. Java, for example, allows one to probe the correctness of any statement in the code by using the assert keyword followed by a correct boolean expression that predicates on the values of interest. If the expression does not hold true, the execution environment raises an assertion exception.

Embedded assertions [15] can be written in the same language as the one used to implement the program or they can be written in another language. In the latter, the code can be written within the language comment marks and an external interpreter is responsible to locate and to execute the assertions.

An issue on the embedded assertion approach may be found if the oracle represents a substantial part of the code. In this case, it can reduce performance when embedded in the code, but removing them after the test phase can cause unexpected problems such as changing the behavior over time, which may be critical for real-time systems. Also, consistency between test oracle specification and the program implementation should be ensured [23], which can be a problem in software maintenance.

Wrappers represent an alternative for embedded assertions. The oracle specification, in this case, does not incur in the code modification. Given a class or component under test, the tester creates a second class with the same interface as the original, but with methods containing contracts to be checked. A test driver communicates with the wrapper that checks if the class under test complies with the specified.

The representation layer of the wrapper is responsible for the conversion of concrete values into abstract values and the abstraction layer compares the abstract values with the post-conditions. The wrapper class overwrites the public methods of the original class. These overwritten methods call the original methods and the test is performed by running the wrapper.

The constraints can be written in an external environment which can be part of a tool set. This tool can have mechanisms to allow one to map the constraints to the implementation and a checker that compares the mapped outputs, or even internal states, with the constraints.

Many approaches we have found by our systematic review are some kind of variation of what we presented in this subsection. Some works focus on different specification languages, others present a known approach with different examples.

For example, Cheon [24] proposes an automated testing approach for Java programs by combining random testing and assertions in OCL. The OCL constraints are translated to runtime checks in AspectJ. The resulting aspect is called constraint checking aspect and it exists separately from the implementation code. The OCL constraints are translated to pointcuts and advices. Pointcuts define execution points and

advices perform constraint checks. The authors cite the Dresden Toolkit [25], which can interpret OCL constraints on a UML model and generates runtime constraints checking code in AspectJ. They suggest adapting this tool for their proposed approach.

Nardi [26] presents a process, a method and a tool for generating oracles for Simulink-like models. TRIO/Apolom, a temporal specification language adapted from TRIO [27], is used to describe the oracle information. Apolom, an oracle generator, allows the definition of an oracle information and the mapping between the Simulink model and the oracle information. It also interprets the oracle information written in TRIO/Apolom and compares it against the results from a Simulink model execution.

The oracle definition process behind its solution, presented in Figure 2, consists of three steps: specification, mapping, and instrumentation.

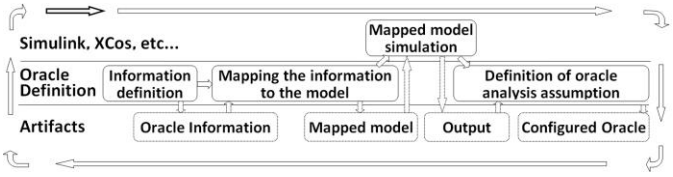

Fig. 2. Oracle definition process. Source: [26]

The *oracle information* states what the expected outcome is and how it should be analyzed. A suitable mapping of the attributes of the oracle specification onto the respective signals of the model results in an instrumented version of the model. The instrumentation inserts a block in each mapped signal to produce log files during the simulation, allowing the oracle to retrieve the data without impacting its behavior. The oracle analysis (off line) states how to evaluate simulation data against the specification.

In any specification-based approach, there are limitations related to the level of information detail. Using constraints, the oracle can report false positives, i.e., it can report a "pass" verdict when the result is actually wrong. But it will not report a false negative (a "fail" when the result is correct), except if the oracle specification is incorrect. For example, let us suppose a program which calculates the sine function and a post-condition (used by the oracle) which determines that any value below -1 and above 1 is incorrect. If the program returns 2 as output, the specification-based oracle will report a "fail" correctly. And for all "fail" reports, the oracle will be always right. However, if the program returns *sin(90)=0.5* as an output, the oracle will report as a "pass" because the output is between -1 and 1, but the result is incorrect (a false positive). Such limitation means that when the oracle states an incorrect output, the verdict is reliable; but when it states that the output is in agreement with the specification, the oracle may not be reliable.

Temporal specifications: on our systematic review, we found a particular context on the use of an oracle: real-time systems, in which time is an important property to be considered. Examples of languages used by oracles that allow the specification of temporal properties are MITL [28], RTIL [1], TRIO [26], timed Petri nets [29] and Lustre [30][7].

Wang [28] presents an approach for automatic generation of oracles for real-time systems based on MITL specification (Metric Interval Temporal Logic). From this specification, one can create a model in timed automata with accepting states (TAAS), which is an automata that has clocks with two attributes: new and old. Their values do not change until some time designation exists in the current state. A sequence of timed states satisfies the specification if it can reach a final state of the automaton built from the specification.

There are also other concepts of time like those presented in the following requirement of a Mars probe landing system: in the event of an error condition, the system must switch to emergency mode. The requirement description uses three variables: *Status*, *TimerInt* and *Done*, as shown below:

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

Their representation in temporal logic is:

$$\square_{[0,\infty]}((TimerInt \wedge \neg Done) \Rightarrow \Diamond_{[0,60]}(Status = Emergency))$$

The square-shaped symbol represents "always in the interval" and the diamond means "sometime in the interval".

An automaton, in the example, was generated automatically based on temporal logic, with 9 states and 20 transitions. Given a timed state sequence, the oracle identifies whether it is correct or not according to the specification.

B. Metamorphic relation based oracles

The concept of metamorphic testing can be understood as follow: "although it may be impossible to know whether the output of an application is correct for a particular input, these applications often exhibit properties such as if an input or system state is modified on a certain way, it can be predicted the new output, given the original output" [31]. A metamorphic relation expresses these properties.

Chen [32] presents a case study of metamorphic relations applied to a sine function. The authors list ten metamorphic relations such as $sin(x) - sin(x+ 2\pi) = 0$. Even without the knowledge about the expected output, it is known that the described relation must be true for any value of x. The tester can create test cases whose inputs are x and $x+2\pi$. Then, if the respective outputs subtraction is different of 0 or outside an accepted precision threshold previously established, an error is revealed.

In the literature, we could find that metamorphic relations can be applied to a broad set of problems. As examples, in an array ordering problem, a metamorphic relation may state that the elements after and before the ordering must be the same. In a program that calculates the shortest path on a graph [32], the metamorphic relation *shortestPath(H, A, C) = shortestPath(H, A, B) + shortestPath(H,B,C)* could be used as oracle information, considering H as a graph, A and C as the origin

and destiny nodes in *H*, and *B* as a node in the shortest path between *A* and *C*.

But comparing outputs can be error-prone for large data sets especially if small variations in the results do not mean error indication or when there is non-determinism in the results. For example, a NP-complete algorithm that finds a path on a graph does not necessarily find the shortest path between two points, but finds an acceptable one. Thus, a metamorphic relation as *shortestPath(A, C)=shortestPath(C, A)* may not be applicable because the algorithm may find two different paths when the origin and destiny are inverted but, even so, the result could be acceptable.

Ding [33] presents an experimental study on a "real-world" program from the Biomedical Laser Laboratory at East Carolina University. Metamorphic testing was applied on an image processing program used to reconstruct 3D structure of biology cells. As example of metamorphic relations, the tester adds mitochondria with different shapes to the cell images so that the 3D structures of these new mitochondria can be built. Then, the original 3D structures should not be changed, and the volume of mitochondria is expected to increase.

Mayer [34] describes an empirical study on metamorphic testing with the use of Java applications that calculates the determinant of a matrix. In conclusion, the authors suggested four guidelines: metamorphic relations that are in the form of equalities are especially weak; if the relation is an equation with linear combinations on each side and at least two terms to one side, then it is not vulnerable to erroneous additions but it is vulnerable to erroneous multiplications; typically good metamorphic relations contain much of the semantics of the software under test; metamorphic relations similar to the strategy used for implementation are limited. Chen [32] gives the following guideline: metamorphic relations that cause higher "difference between executions" tend to be better. But this concept was not explicitly set. More research should be conducted to provide broader and detailed guidelines.

Zhang [35] presents an experiment with 3 programs: *Boyer*, which returns the index of first occurrence of a pattern in a string; *BooleanExpression*, which validates boolean expressions; and *TxnTableSorter*, an office application. The subject participants were 38 postgraduate students enrolled in an advanced software testing course. The authors investigated (i) if the students could appropriately apply metamorphic testing after being trained; (ii) if they could identify correctly and usefully metamorphic relations to the target program; and (iii) if the same metamorphic relation could be discovered by multiple students. For all these questions, the answer was yes. They also investigated what is the effort in terms of cost, in applying metamorphic testing. According to the results, metamorphic testing has the potential to detect more failures than assertion checking. On the other hand, may be less efficient in terms of cost.

In general, students identified a greater number of assertions than metamorphic relations. The number of metamorphic relations and assertions found varied significantly among students. The authors believe that metamorphic testing helps developers to increase the level of abstraction better than assertions.

Oracles based on metamorphic testing rely on relations that are specific to the system domain and may not be evident to be found. According to Chan [36], the choice of metamorphic relations was based on experience of the testers. All the works we have found present study case of simple programs or functions. Complex systems, with different functionalities would require different sets of metamorphic relations. Supporting that observation, Murphy [37] notes that metamorphic testing can be a manually intensive technique for more complex cases.

The reliability of a metamorphic relation based oracle is the same as the specification based oracle.

C. Machine learning based oracles

Machine Learning, as neural networks, has the capacity to simulate a software behavior based on the input/output pairs [38]. They can be used as continuous [39] or discrete [40] function approximators and that property can be explored to build oracles. We have found the following neural networks used as oracles: backpropagation, RBF (Radial Basis Function) and SOM (Self-Organizing Map).

There are two kinds of operation procedures in a neural network use: training and regression (or association, if the network is used as as classifier [39]. Given a training set composed by input/output pairs, a neural network (in the role of continuous function approximator) is capable to find an approximated function of an deterministic computational process. Once trained, the network can generate, at the regression operation, the expected outputs to input data that are not part of the training set. For a neural network used as a discrete function approximator, it can be trained with a set of pairs input/output, where the output is a category to the input. At the association operation, the network can classify other inputs in one of the given categories.

As example, Aggarwal [40], Chan [36] and Jin [39] address the use of neural networks as oracles in problems involving classification. Two of these papers present as a case study, an oracle for triangle classification into isosceles, scalene and non-equilateral. The inputs are three integers that represent the length sides of a triangle. The output is the classification in equilateral, isosceles, scalene or not a triangle. It is given, as training set, correct input/output pairs. After trained, the neural network is able to classify new inputs into the presented categories. But a neural network may misclassify the inputs.

According to Vanmali [41], neural network can be justified for a variety of reasons, such as situations where the original version is unavailable. There may be occasions where the input and output data from the original program are not critical and the application of a neural network instead of the original program may save computer resources.

Shahamiri [38] presented an experiment with a registration-verifier program. Based on the student records, the program validates the registration, decides the maximum courses students can select and if a discount is applicable or not. A

backpropagation neural network was used as an oracle. The authors evaluated the oracle with a golden version of the test cases and mutated test cases. Comparing the results between the golden version and the oracle outputs and the results between the oracle outputs and the mutated test cases, the authors measured the total of true positives, true negatives, false positives and false negatives. The oracle accuracy was of *95.37%*.

Neural networks are not the only machine learning resource. Wang et al.[42] apply support vector machine as supervised learning algorithm (SLA) to test reactive systems. In a first step, user guidance or assertions can be used to collect verdicts to test traces. Such traces are converted into feature vectors to rain the SLA, which is used as test oracle. In the experiment, statements are inserted into the software under test (SUT) to collect the verdicts. Also, bugs were implemented into the SUTs to check the correct verdicts of the test oracle. The results show that the proposed technique incurs little burden and overhead. The training time varies between 5 and 42 seconds and the testing time varies between 2 and 29 seconds. The correct verdicts fluctuate between *92.88%* and *96.52%*.

There are limitations on the use of machine learning as oracle. They can report false positives and false negatives. This means that if it states that the output is incorrect, the oracle may be wrong as much as if it states that the output is in agreement with the specification. In both cases, the tester must check if the oracle verdict is correct. In case of false negatives, a tester will spend time searching for an error that does not exist.

The neural network accuracy and precision is related with the comparator threshold [38]. This property defines the oracle precision and it represents an interval. If the difference between the expected output and the obtained output is inside this interval, the oracle should report a "pass". Otherwise, the test failed. With higher threshold (higher interval acceptance), an oracle may present more false positives. With lower threshold, the comparison may be more precise but they may report more false negatives and may lose accuracy.

Because neural network are approximators, the tester must be aware about the uncertainty characteristic of such oracles. Their reliability may reside on statistical data, as expected accuracy based on previous similar case studies. But when using these oracles on new domains, the tester may not have parameters to decide about the oracle reliability.

Other limitations are observed by Jin [39]: the input data may not be easily represented by neural networks, as characters and strings. Still, different elements in the input vector may have unequal contribution to the network. Deciding the structure of the network may not be easy, as the amount of layers and neurons. How to select the training set from the test cases is another key problem that must be considered carefully.

After our systematic review we noted a lack of studies in the area. Most papers focus on the ANN viability as oracles, but more research need to be done as comparisons between oracles, the influence of different weights, bias, activation functions, layers and topologies. For instance, Lu [43] conclude that the use of RBF is feasible as an oracle, and besides the evident contribution, there is no comparison with other neural networks.

D. N-version and similars

N-version is based on several implementations of the program, developed independently, and with the same functionality of the software under test [2]. These versions are used as pseudo-oracles. If there is disagreement about the output in the versions, the decision is based on voting and the most common values are used as the expected outputs. As example, Shimeall [44] use the n-version concept on programs written in Pascal from a specification for a problem of combat simulation.

Another approach, a variation of n-version, is m-mp. In this approach, just the core functionality, or those that are critical, is implemented as an oracle. It provides low cost based on the justification that the program model do not need to be equivalent to the main program [45].

The idea of comparing results between two or more implementation can be extended to programs that already exist. A golden version of a program can be used as an oracle, for example in regression testing, component harvesting [46] or "Multiple-implementation Testing" (MiT) [47].

Tsai et al. [48][49] propose a technique of majority voting to test a large number of Web Services (WS) that already exist and belong to a single specification.

Hummel [46] propose the creation of oracles from the same basic technologies that can be used to find components for reuse (such as *Extreme Harvesting*). Thus, it uses the components found in the Internet searches combined as a pseudo-oracle to measure the confidence of the built components.

A few limitations on oracles based in n-version can be quoted. When a system is stable, it can be used as oracle to test new versions, but just for the old functionalities. If new characteristics are added, they will need other kind of oracle. This not means that they are useless, but it means that other oracles should be used to test the additions.

In n-version, the approach requires multiple implementations of the system and, consequently, it has high cost. Besides, the oracle is not reliable because it can have errors, as the program under test. The difference is that, using different teams and approaches, it is less probable the both versions contain the same defects. If so, it may be unlikely that the oracle and the program generate the same wrong output, which means that it is improbable that the comparator report that the program passed the test even if it should report as a fail (false positives).

But, because the tester must verify if it is the oracle or the program under test that is defective, an extra time is needed and it may influence on the cost.

Shimeall [44] mentions that n-version is not a substitute for functional tests. In the experiment, n-version did not tolerate many of the failures detected by other techniques to eliminate failure.

TABLE 1. Comparative table between different oracle categories

Category	Limitations	False negative	False positive	Practical appliance	Complexity to apply	Non-determinism
Specification	If the specification is wrong, the test is pointless A complete and consistent specification can be almost as complex as the implementation May be a missing concept between the specification layer and the implementation layer The specification must be mapped to the implementation	If the specification is wrong	Yes	Found	Relies on the specification detailing level	May be easy to handle
Metamorphic relation	The metamorphic relation may not be trivial to find and resides in the tester experience Same limitations as specification-based oracles	If the metamorphic relation is wrong	Yes	Not found	Relies on the tester experience to define the metamorphic relations	Hard to handle
Machine learning	They are function approximators. The network precision depends on many variables and may not be easy to achieve.	Yes	Yes	Not found	Relies on how difficult is to prepare the input set	Hard to handle
N-version and similar	It can be very costly A golden version may not be available There is no guarantee that the version is error free	Yes	Yes	Found	If a golden version is available, it is usually easy to apply	Hard to handle

V. DISCUSSION

Table 1 presents a comparison among the oracle categories and their limitations. For each category, it is presented a resume of its limitations, if the oracle generates false negatives and false positives, if practical application in the industry was found (excluded the experimental studies with industrial examples), and how an oracle handles non-deterministic results. The summary is composed in order to answer the following questions:

- Which limitations has an oracle?
- How much an oracle can be precise?
- Is an oracle applied on “real-world” systems?
- Is an oracle costly?

These questions may help the tester to decide what oracles are suitable or worth of being considered.

We believe the oracle categories are complementary and their integration can improve the capability to find errors. For example, if there is a set of well defined rules that must be followed and the system has a golden version, both specification-based and n-version-like oracles could be used. If a system is a classifier and there is a set of well defined rules, neural networks and specification-based oracles could be used.

In our research, we found that most works focus on specification-based oracles. This category and n-version are found since late 80's while machine learning and metamorphic testing as oracles are found after year 2000.

Some oracles are notably useful on a specific context as regression testing and when a golden version is available.

Other oracles, as neural networks and metamorphic testing based oracles, even promising, need more study.

We have found that most works use simple examples and we did not find many papers presenting experiments in the industry or ``real-world'' systems. This is a gap that we found when conducting the systematic review. Possibly, it is a shortage on Industry feedback or a lack of practical studies. Just a few approaches were applied to “real-world” systems (Leto and T-Vec, for instance) and their use means changes in the development process that some Companies may not be inclined to deal. We could not find an oracle specification language, environment or method that could be easily implanted. Such observation led us to tackle such gap and we defined a process, method and tool which allows the generation of test oracles for Simulink-like models [26][50]. To note [51]: (i) the oracle information can be almost as complex as the program under testing and must be checked carefully. Supporting this difficulty, Kim et al. [52] cite the trade-off between the specification precision and the simplicity; (ii) the test harness (or the oracle procedure) is not a trivial program and should be checked carefully.

We cite a few promising fields to explore, as the research on easier ways to apply specification languages, which must be executable to allow its implementation, and that allow one to describe the specification on different abstraction layers. A search for specification languages with high usability has been

increased in the last years [13]. It can be noticed, in the literature, a recent preference by specification languages that are of common use in the industry and academic environments or that have a well known paradigm, namely java assertions, OCL, JML, state-machines and VDM. This former is used in the industry. Many attempts to use such languages focus on specific domains, as java programs, executable UML and log verification. For practical and disseminated application of automated test oracles, it is expected the adoption of specifications with higher usability, which leads to studies about specification language usability.

User-friendly oracle environments may have the same importance of the specification language when considering the application in "real-world" scenarios. An oracle environment should allow the tester to describe the oracle specification in different levels of abstraction and as much complete as it is needed, from a few constraints to an entire and complete system specification. Also, the integration of an oracle environment with different languages and platforms could be explored. Only a few works, as in [26], have tackled such issues.

Another field to explore includes the oracle development processes and methods. As discussed, there is a trade-off between choosing a system specification as an oracle specification or the creation of a separated one. Also, it may not be trivial to track and maintaining the updates between the system specification and oracle specification. We believe that more research should be done on tools that provide such mechanisms.

VI. Final remarks

The importance of testing activity is widely known. However, the difficulty in deciding whether a result is acceptable or not (also known as the oracle problem) hampers such activity.

The comparison between expected output and obtained output is oftentimes executed manually by the tester, which is usually slow, error-prone, and very expensive. Several approaches have been proposed in order to soften the oracle problem and automate the process.

The main contribution of this paper is presenting a critical overview on test oracles, as well as its limitations. We also highlighted possible fields yet to be explored. We aim to provide a guideline to researchers and testers who seek to soften the oracle problem.

References

[1] D. Richardson, S. Aha, and T. O'Malley, "Specification-based test oracles for reactive systems," in International Conference on Software Engineering., 1992, pp. 105–118.

[2] S. Shahamiri, W. Kadir, and S. Mohd-Hashim, "A comparative study on automated software test oracle methods," in ICSEA '09: Fourth International Conference on Software Engineering Advances., Sept. 2009, pp. 140–145.

[3] M. C. Gaudel, "Testing can be formal, too," in Lecture Notes in Computer Science, vol. 915. Springer-Verlag, 1995, pp. 82–96.

[4] E. J. Weyuker, "On testing non-testable programs," The Computer Journal, vol. 25, no. 4, pp. 465–470, 1982.

[5] Y. Mao, F. Boqin, Z. Li, and L. Yao, "Automated test oracle based on neural networks," in Cognitive Informatics, 2006. ICCI 2006. 5th IEEE International Conference on, vol. 1, July 2006, pp. 517–522.

[6] A. Memon, I. Banerjee, and A. Nagarajan, "What test oracle should i use for effective gui testing?" in Proceedings of the 18th IEEE International Conference on Automated Software Engineering., Oct. 2003, pp. 164– 173.

[7] G. Durrieu, H. Waeselynck, and V. Wiels, "Leto - a lutre-based test or-acle for airbus critical systems," Formal Methods for Industrial Critical Systems: 13th International Workshop, FMICS 2008, 2008.

[8] M. D. Davis and E. J. Weyuker, "Pseudo-oracles for non-testable programs," in ACM '81: Proceedings of the ACM '81 conference. New York, NY, USA: ACM, 1981, pp. 254–257.

[9] D. Kim-Park, C. de la Riva, and J. Tuya, "A partial test oracle for xml query testing," in Testing: Academic and Industrial Conference - Practice and Research Techniques, 2009. TAIC PART '09., sept. 2009, pp. 13 –20.

[10] A. Pasala, S. Rao, A. Gupta, and S. Gunturu, "On the validation of api execution-sequence to assess the correctness of application upon cots upgrades deployment," in ICCBSS '07: Sixth International IEEE Con-ference o Commercial-off-the-Shelf (COTS)-Based Software Systems., march 2007, pp. 225–232.

[11] R. Shukla, D. Carrington, and P. Strooper, "A passive test oracle using a component's api," in Software Engineering Conference, 2005. APSEC '05. 12th Asia-Pacific, Dec. 2005, pp. 7

[12] P. A. Nardi and M. E. Delamaro, "Test oracles associated with dynamical system models," Universidade de So Paulo/So Carlos - ICMC, Tech. Rep., 2011.

[13] R. A. Oliveira, U. Kanewala, and P. A. Nardi, "Chapter three - automated test oracles: State of the art, taxonomies, and trends," ser. Advances in Computers, A. Memon, Ed. Elsevier, 2014, vol. 95, pp. 113 – 199. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B9780128001608000036

[14] S. Baharom and Z. Shukur, "Utilizing an abstraction relation document in grey-box testing approach," in ICEEI '09: International Conference on Electrical Engineering and Informatics., vol. 01, Aug. 2009, pp. 304–308.

[15] L. Baresi and M. Young, "Test oracles," University of Oregon, Dept. of Computer and Information Science, Eugene, Oregon, U.S.A., Tech-nical Report CIS-TR-01-02, August 2001, http://www.cs.uoregon.edu/ michal/pubs/oracles.html.

[16] J. Hakansson, B. Jonsson, and O. Lundqvist, "Generating online test oracles from temporal logic specifications," International Journal on Software Tools for Technology Transfer (STTT), vol. 4, no. 4, pp. 456–471, Aug. 2003. [Online]. Available: http://dx.doi.org/10.1007/ s10009-003-0107-8

[17] L. Lazic´ and D. Velasevi˘c,´ "Applying simulation and design of experiments to the embedded software testing process: Research articles," Softw. Test. Verif. Reliab., vol. 14, no. 4, pp.

257–282, Dec. 2004. [Online]. Available: http://dx.doi.org/10.1002/stvr.v14:4

[18] M. Veanes, C. Campbell, W. Grieskamp, W. Schulte, N. Tillmann, and L. Nachmanson, "Model-based testing of object-oriented reactive systems with spec explorer," in Formal Methods and Testing, ser. Lecture Notes in Computer Science, R. Hierons, J. Bowen, and M. Harman, Eds. Springer Berlin / Heidelberg, 2008, vol. 4949, pp. 39–76.

[19] J. Andrews and Y. Zhang, "General test result checking with log file analysis," IEEE Transactions on Software Engineering, vol. 29, no. 7, pp. 634–648, July 2003.

[20] D. Tu, R. Chen, Z. Du, and Y. Liu, "A method of log file analysis for test oracle," in International Conference on Scalable Computing and Communications; Eighth International Conference on Embedded Computing. SCALCOM-EMBEDDEDCOM'09., Sept. 2009, pp. 351– 354.

[21] J. Guttag, "Abstract data types and the development of data structures," Commun. ACM, vol. 20, no. 6, pp. 396–404, 1977.

[22] D. Peters and D. L. Parnas, "Generating a test oracle from program documentation: work in progress," in ISSTA '94: Proceedings of the 1994 ACM SIGSOFT international symposium on Software testing and analysis. New York, NY, USA: ACM, 1994, pp. 58–65.

[23] J. Bieman and H. Yin, "Designing for software testability using auto-mated oracles," in Test Conference, 1992. Proceedings., International, Sep 1992, pp. 900–.

[24] Y. Cheon and C. Avila, "Automating java program testing using ocl and aspectj," in Seventh International Conference on Information Technol-ogy: New Generations (ITNG)., april 2010, pp. 1020 –1025.

[25] B. Demuth and C. Wilke, "Model and object verification by using dresden ocl," in Russian-German Workshop ¨Innovation Information Technologies: Theory & Practice¨. Ufa, Russia, July 2009, pp. 1020 – 1025.

[26] P. A. Nardi, "On test oracles for simulink-like models," Ph.D. dissertation, Instituto de Ciências Matemáticas e de Computação, São Carlos, Brasil, 2013.

[27] C. Ghezzi, D. Mandrioli, and A. Morzenti, "Trio: A logic language for executable specifications of real-time systems," J. Syst. Softw., vol. 12, May 1990.

[28] X. Wang, Zhi-Chang, and Q. S. Li, "An optimized method for automatic test oracle generation from real-time specification," in ICECCS 2005: Proceedings of the 10th IEEE International Conference on Engineering of Complex Computer Systems., June 2005, pp. 440–449.

[29] J.-C. Lin and I. Ho, "A new perspective on formal testing method for real-time software," in Euromicro Conference, 2000. Proceedings of the 26th, vol. 2, 2000, pp. 270–276 vol.2.

[30] J. Bouchet, L. Madani, L. Nigay, C. Oriat, and I. Parissis, "Formal testing of multimodal interactive systems," Engineering Interactive Systems: EIS 2007 Joint Working Conferences, EHCI 2007, DSV-IS 2007, HCSE 2007, Salamanca, Spain, March 22-24, 2007. Selected Papers, pp. 36– 52, 2008.

[31] C. Murphy, "Using runtime testing to detect defects in applications with-out test oracles," in FSEDS '08: Proceedings of the 2008 Foundations of Software Engineering Doctoral Symposium. New York, NY, USA: ACM, 2008, pp. 21–24.

[32] T. Chen, F.-C. Kuo, T. Tse, and Z. Q. Zhou, "Metamorphic testing and beyond," in Software Technology and Engineering Practice, 2003. Eleventh Annual International Workshop on, Sept. 2003, pp. 94–100.

[33] J. Ding, T. Wu, J. Lu, and X.-H. Hu, "Self-checked metamorphic testing of an image processing program," in Fourth International Conference on Secure Software Integration and Reliability Improvement (SSIRI)., June 2010, pp. 190 –197.

[34] J. Mayer and R. Guderlei, "An empirical study on the selection of good metamorphic relations," in Computer Software and Applications Conference, 2006. COMPSAC '06. 30th Annual International, vol. 1, Sept. 2006, pp. 475–484.

[35] Z.-Y. Zhang, W. Chan, T. Tse, and P. Hu, "Experimental study to compare the use of metamorphic testing and assertion checking," Ruan Jian Xue Bao/Journal of Software, vol. 20, no. 10, pp. 2637–2654, 2009, cited By (since 1996) 0.

[36] W. K. Chan, M. Y. Cheng, S. C. Cheung, and T. H. Tse, "Automatic goal-oriented classification of failure behaviors for testing xml-based multimedia software applications: an experimental case study," J. Syst. Softw., vol. 79, no. 5, pp. 602–612, 2006.

[37] C. Murphy, K. Shen, and G. Kaiser, "Using jml runtime assertion checking to automate metamorphic testing in applications without test oracles," in ICST '09: International Conference on Software Testing Verification and Validation., April 2009, pp. 436–445.

[38] S. Shahamiri, W. Wan Kadir, and S. Ibrahim, "A single-network ann-based oracle to verify logical software modules," in 2nd International Conference on Software Technology and Engineering (ICSTE)., vol. 2, oct 2010, pp. V2–272 –V2–276.

[39] H. Jin, Y. Wang, N.-W. Chen, Z.-J. Gou, and S. Wang, "Artificial neural network for automatic test oracles generation," in Computer Science and Software Engineering, 2008 International Conference on, vol. 2, Dec. 2008, pp. 727–730.

[40] K. K. Aggarwal, Y. Singh, A. Kaur, and O. P. Sangwan, "A neural net based approach to test oracle," SIGSOFT Softw. Eng. Notes, vol. 29, no. 3, pp. 1–6, 2004.

[41] M. Vanmali, M. Last, and A. Kandel, "Using a neural network in the software testing process," International Journal of Intelligent Systems, vol. 17, no. 1, pp. 45–62, 2002, cited By (since 1996) 8.

[42] F. Wang, L.-W. Yao, and J.-H. Wu, "Intelligent test oracle construction for reactive systems without explicit specifications," in Dependable, Au-tonomic and Secure Computing (DASC), 2011 IEEE Ninth International Conference on, dec. 2011, pp. 89 –96.

[43] Y. Lu and M. Ye, "Oracle model based on rbf neural networks for automated software testing," Information Technology Journal, vol. 6, no. 3, pp. 469–474, 2007, cited By (since 1996) 1.

[44] T. Shimeall and N. Leveson, "An empirical comparison of software fault tolerance and fault elimination," in Software Testing, Verification, and Analysis, 1988., Proceedings of the Second Workshop on, Jul 1988, pp. 180–187.

[45] L. Manolache and D. Kourie, "Software testing using model programs", Software - Practice and Experience, vol. 31, no. 13, pp. 1211–1236, 2001, cited By (since 1996) 2.

[46] O. Hummel and C. Atkinson, "Automated harvesting of test oracles for reliability testing," in Computer Software and Applications Conference, 2005. COMPSAC 2005. 29th Annual International, vol. 2, July 2005. Pp. 296-202. Vol. 1.

[47] K. Taneja, N. Li, M. R. Marri, T. Xie, and N. Tillmann, "Mitv: multiple-implementation testing of user-input validators for web applications," in Proceedings of the IEEE/ACM international conference on Automated software engineering, ser. ASE '10. New York, NY, USA: ACM, 2010.

[48] W.-T. Tsai, Y. Chen, D. Zhang, and H. Huang, "Voting multi-dimensional data with deviations for web services under group testing," in 25th IEEE International Conference on Distributed Computing Sys-tems Workshops., June 2005, pp. 65–71.

[49] W.-T. Tsai, Y. Chen, R. Paul, H. Huang, X. Zhou, and X. Wei, "Adaptive testing, oracle generation, and test case ranking for web services," in Computer Software and Applications Conference, 2005. COMPSAC 2005. 29th Annual International, vol. 1, July 2005, pp. 101–106 Vol. 2.

[50] P. Nardi, M. Delamaro, and L. Baresi, "Specifying automated oracles for simulink models," in Embedded and Real-Time Computing Systems and Applications (RTCSA), 2013 IEEE 19th International Conference on, Aug 2013, pp. 330–333.

[51] D. Peters and D. Parnas, "Using test oracles generated from program documentation," IEEE Transactions on Software Engineering, vol. 24, no. 3, pp. 161–173, Mar 1998.

[52] D. S. Kim-Park, C. de la Riva, and J. Tuya, "An automated test oracle for xml processing programs," in Proceedings of the First International Workshop on Software Test Output Validation, ser. STOV '10. New York, NY, USA: ACM, 2010, pp. 5–12.