Strategies of Automated Test Oracle – A Survey

Kamaraj Kanagaraj and Arvind Chakrapani

Kamaraj Kanagaraj, Assistant Professor, Department of Information Technology, Dhanalakshmi Srinivasan College of Engineering, Coimbatore.
Arvind Chakrapani, Associate Professor, Department of Electronics and Communication Engineering, Karpagam College of Engineering, Coimbatore.

Received 18 September 2016; Accepted 15 January 2017; Available online 29 January 2017

Address For Correspondence:
Arvind Chakrapani, Anna University, Department of Electronics and Communication Engineering (PG), Karpagam College of Engineering, Coimbatore, Tamil Nadu, India

Copyright © 2017 by authors and American-Eurasian Network for Scientific Information (AENSI Publication). This work is licensed under the Creative Commons Attribution International License (CC BY).

ABSTRACT
Software testing is an important activity to confirm the system under test (SUT) meets the specification. Testing the software consumes nearly 50% of the overall project cost. Test Oracle is a process to check the correct behavior of the SUT. Automating the testing process includes generation of the test cases, comparison of the behavior observed with the expected behavior of the system and evaluation of the result for acceptance. Completely automating the test oracle is a difficult task to achieve. The main objective of this article is to present a detailed review on the current research trends in automated test oracle generation and various techniques based on the requirement and design specifications of the SUT, before comparing the observed behavior, the test cases must be reduced so that the cost and effort are minimized. An effort also has been made to categorize test oracle based on its characteristics. Measuring the 'Test Oracle' enables the design of a complete and accurate oracle. A comparative analysis is made based on the automated test oracle which answers the research query on the contribution of the test oracle methods, cast and effectiveness of the methods and accuracy level of the methods used to generate test oracle. Based on the study it is evident that there a need of constructing new tools to generate test oracle automatically in order to reduce the cost, time and effort involved in generating based on the nature of the SUT.

KEYWORDS: Test Oracle, Requirement Specification, Design Specification, System under test (SUT)

INTRODUCTION

Test Oracle Problem:
Test oracle is a mechanism to check whether the software under test results in correct behavior or not. Test oracle automation is a process which generates the possible number of suitable test case to test the SUT and automatically compares the result with the expected behavior of the system. The test oracle automation is still comparatively a less solved problem [1]. The main challenges behind the test oracle automation is to choose the appropriate method for generating the test case, at what time the method has to be applied during the software development process, choosing a suitable method for the SUT based on the application area of the software [2]. The test case is generated from the specification of the software which is in the form of formal specification method, relation notation or state oriented notation; design of the software, exception condition [3] and also from the source code [4].

Different reasons to automate the Test Oracle processes in software testing are
**Human Oracle Cost problem:**

The effort of reducing the cost involved in writing the test oracle and evaluating the test outcomes is referred as human oracle cost [5]. This led to focus on quantitative and qualitative reduction in the testing process.

**Role of Tester’s Knowledge:**

The experience and comprehensive knowledge of the tester in identifying and categorizing the failure is considered to be significant in the testing process. Knowledge includes domain knowledge, system knowledge and generic software engineering knowledge [6].

1) Efficiency – with respect to the skill set of the tester who generates test oracle.
2) Correctness – the percentage of deviation in the creation of the test oracle by human effort is comparatively high.
3) Reliability – manual test oracle is less reliable than the output generated from automated test process.

**Testing time:**

The amount of data that is been generated during the process is large which is difficult for the tester to observe and analyze in a stipulated time [7].

The Test Oracle is conceptually viewed as combined process of generating, comparing and evaluating the test case to verify the specified requirement. Vineeta *et al*. classified the test oracle approach based on the specification and documentation, heuristic, consistency, statistical and model based oracle [8]. Figure 1 shows the classification of test oracle strategies. With reference to approach employed in test oracle strategies, the next section categorizes the automated test oracle based on the test case generation and validation of the SUT.

![Fig. 1: Test Oracle Strategies](image-url)

3 **Specification based Oracle:**

In Specification based oracle, test oracle is specified using one of the specification languages to verify the behavior of the product to the desired specification. The specification language takes several forms which are model based language, assertions and contracts, state transition system and algebraic specification.

3.1 **Test Oracle from Program Documentation:**

Peters and Parnas [9], design document is precisely defined for the software system to improve the quality. The design document which uses a relational form (tabular expression) is much precise and readable than other types of representation [10]. Mathematical expression represented in a tabular form can be easily used to derive the test oracle. A Test Oracle Generator (TOG) tools is developed to produce the test oracle which is precise and readable, has minimum stated requirement and written in expressive notation. The author has used LD-relations
for the specification. As stated by Parnas et al. [11], the design document of a software system consists of system requirement document, system design document, software behavior specification, software module design, module interface specification and module internal design document. The approach suggested uses the module internal design document to generate the test oracle. The module internal design document describes the data structures used in the module of the software system, the interpretation of that data structure and the function call. A TOG tool is an added version of the tool designed in Peter et al. [12] work. In the new approach, the test oracle is generated which is of C language procedure with some C++ objects and an optimization technique is used for minimizing the time taken to generate the test oracle. Difficulty arises when the software system specification is not correctly defined which causes certain difficulty in the generation of good test oracle. The above limitation can be avoided by careful inspection and employing specification checking tools. Other limitations include the assess ability of the data structure by the oracle and the harness of the program document.

3.2 Test Oracles based on Algebraic Specifications:

Algebraic specification has emerged as a formal method of specification in early 1980’s [13]. The early work [14] implemented in a procedural programming language employs an algebraic specification which is based on the two terms of the given signature. The terms represent the sequence of call and the value. Study of Doong et al. [15] on the effectiveness of object oriented program testing based on the algebraic specification is the method that extends the test cases by including negative test cases, which results in a non-equivalent result. Gaudel et al. [16, 17] developed an approach based on the algebraic specification in which the method uses an observation context and that is used for the automatic comparison of the structured value. The approach proposed by Hong Zhu is based on the observable sort by dividing the equation into groups by introducing a pre order relation to match the relation between classes [18]. The validity of the observed work states that a well structured algebraic specification will be able to produce a good oracle.

3.3 Z specification for Oracle generation:

McDonald and Strooper approach uses a Z specification to generate a passive C++ Test Oracle in which the test oracle is classified as active test oracle and passive test oracle [19]. The difference between active and the passive is that active implements the expected behavior of the SUT where as passive test oracle verifies the behavior of the implementation. A translation method adopted from [20, 21], where the object specification is translated into a C++ code. The translation is carried out through two steps – optimization and the oracle translation. In optimization, the specification is translated into a systematic form (implementation language) while in the oracle translation, the systematic form is translated into a passive oracle of C++. A GMD cocktail Compiler Generation Toolbox [22] is used to generate the Warlock tool, a prototype oracle generation tool. Specification from the source file is read by the Warlock tool and expressed as standard LATEX object-Z macro package. The tool is composed of modules and flow of data from one module to the other using arrows. The first module scans the input from the source file as streams of character and produces number of lexical tokens. The data structure to store and manipulate the specification is provided by the abstract syntax tree, where the tree is represented as a hierarchy of nodes. The attribute evaluator and the type checker are used to traverse the tree and decide whether the specification satisfies the type rule of object-Z language. The code generator translates the abstract tree to generate the test oracle interface. Figure 2 shows the implementation of the oracle generation using warlock tool from the Z specification for the SUT. The user defined function and the fitness function within the optimized specification are the main limitations of the method.
3.4 Test Oracle based on mathematical notation:

Formal specification of representing the software system is advantageous than the informal representation of the system. Alawneh and Peters [23] formal specification uses a mathematical notation for generating the test oracle automatically. The approach derives the test oracle from the module internal design document. Open source libraries are used to build a specification editor in the eclipse framework to support the production of the document which shows many views of the source file. A complete specification for the expected behavior of the system is first written in a tabular form that contains the definition of the function. Next step in the process is to generate the test oracle by taking the program under test and oracle procedure as the input to the function defined in the framework. The significant advantage of using the JUnit tool includes the ability to understand the test, write new test oracle, and also provides a user-friendly graphical user interface (GUI). From the scalar and the tabular expression, the tool can creates test oracle.

3.5 Generating oracle from your favorite Temporal Logic Specification:

Dillon and Ramakrishna [24] proposed a method which uses a tableau algorithm that produces test oracle from the temporal logical specification. The tableau algorithm is initiated with the rule that expresses the semantic of the temporal logical specification which produces the tableaux from the specification. A finite state automaton is described form the tableaux generated. The tableau algorithm is parameterized to accept the variety of temporal operation and which also helps for fine-tuning the algorithm and then the oracle produce a meaningful feedback about the fault if any. The oracle is used for run-time monitoring and debugging in addition to the post checking of the logged trace since the error is traced as early as possible.

3.6 Test Oracle from formal specification:

Day and Gannon [25] described a model of generating test oracle from the formal specification. A simple version of Pascal, CF Pascal is used to specify the system. A syntax section and semantic section which is used to construct the oracle, form the specification is shown in the figure 3. Two grammars are used in the syntax section to specify the input and output file while the semantic section describes the relationship between input and output. Standard semantic rules are defined for operating the sequence and a special-purpose function is also defined to operate the textual sequence. The syntax section and the semantic section are compiled to produce an oracle to check the consistency of the output.
3.7 A Framework for Specification-Based Testing:

Stocks and Carrington [26] presented a formal method to generate the test oracle by using the Z notation for the specification. Specification represents the system behavior and is considered as an important tool for the generation of the test oracle from which the expected test result can be derived [30]. The specification precisely describes the relationship between the input and the output states which is considered as test oracle. The oracle generated is a simple model of oracle but it is flexible and can be easily extended. The comparison of the actual output to the pre-calculated output is performed in order to validate the input. The output space of the operation represents the oracle template for each data, it describes the data sets and input – output relationship.

Fig. 3: Oracle form formal specification

3.8 Data Mining Algorithm used to generate Test Oracle:

Last et al. [27] proposed a model based on the data mining approach for automated modeling of the SUT. An info-fuzzy network is used to represent each output variable of the SUT, a network induction algorithm constructs the network for each output. The root node represents an empty set of input attribute and combination of the input attributes are represented by the leaves in the tree, where the info fuzzy network (IFN) looks like a tree. During the training phase of the network the input and the output is given to the Random Test Generator (RTG) so that the test cases are generated. The test cases are given as input to the test bed and the info logic algorithm, where the test bed gets the input from the legacy system. The IFN modules for each output variables are generated and information about the logical rules, test cases and the equivalence class are derived. The approach is applied and result shows that the technique is used to model complex software systems and automatically produces the relationship existing in the software by creating a non redundant test case.

4 Heuristic based Oracle:

The test oracle in which a selected set of test condition is checked for the SUT and a simpler algorithm or a consistency check based on the heuristic can be used to check the remaining values is referred as “Heuristic based Oracle”. This type of oracle has the advantage of easy implementation, fast execution and wider fault coverage, which is implemented using a heuristic algorithm [28].

4.1 Automatic Test Oracle generation using ANN:

Hu Jin et al. [29] designed a test oracle by introducing Artificial Neural Network (ANN) and applied the same to validate and evaluate the test oracle. The underlying concept of using ANN is training the network and also for function approximation. A back propagation algorithm is used to construct the test oracle and training set is framed to supervise the learning. The test cases are trained by ANN and verified. The next step in the process is to generate the test oracle and evaluate its correctness. If the model doesn’t match, the test case is reconsidered. Since the input data may be represented in character, string or other forms, the input data must be mapped to the input pace of the ANN. The next major issue in using the ANN for oracle generation is the decision of ANN, where a careful consideration should be followed to select the type of ANN so that the appropriate training set is selected. The designed model is tested with TRIANGLE classifying program to conclude that ANN is a reliable model in designing heuristics base oracle.

4.2 Neural Networks for Automated Test Oracle:

Ye et al. [30] with the help of Neural Network developed an automated test oracle to describe the relation between input and output as a function. A continuous function is investigated, in which when the input or output is not a real number, they can be coded into real number. Automated test oracle can be generated and compared with the expected output, which is an approximate output since it is equal to the expected output in any precision. The neural network is used as a tool to generate the approximate output and it has two paradigms in general, namely supervised and unsupervised [31]. Initially the neural network is constructed with precision
degree ε and the weight is initialized. Step 1 of figure 4 shows the construction and the initialization of the network with the weight and precision degree set. The second step in the process of oracle generation is training of the neural network, step 2 of figure 4. The back propagation algorithm is used to train the network with the training set and once the stopping criterion is obtained it accepts the input and provides it to the neural network. The approximated output is generated and the expected output is obtained from the application under test (AUT) and compared to determine its correctness. Figure 4 shows the generation of test oracle from the neural network for all input and for different precision degree. In this approach all the outputs are not generated, hence it reduces the time and labor cost of testing.

![Fig. 4: Test oracle from ANN](image)

### 4.3 Automated Test Oracle with Test Decision Making Structures:

Shahamiri et al [32] addressed the issue of test decision making by designing an automated test oracle. A simulated model of the SUT is required to generate automatically the output for all the given input. A dataset is used to train the model, pre condition for the training is that all non numeric values should be converted to numeric values. The trained network acts as an oracle and generates output from the input provided to the network for the logical unit under test. Then the output is compared with the desired output by a comparator. The difference in this approach to the previous approaches is that the training dataset given as an input to the ANN is the normalized dataset. The approach is applied to a registration-verification application and verified with 950 training sets and different network architecture models are also used for testing purpose. Mutation testing is performed on the proposed approach with 26 faults, in which 98% of the mutant is identified in the SUT.
4.4 BPAS and ATCGS:

Sofokleous and Andreou [33] designed a framework for dynamic test data generation. The framework contains a program analyzer which takes care of parsing the source code, representing the source code by control flow graph, extracts variables and paths, and evaluate automatically. The second part of the framework is the test case generation system which has two algorithms, one algorithm to evolve the test case and the other is to uncover the statements and conditions in the specific path. The program analyzer performs the static analysis of the SUT by the runtime system specified and also performs the dynamic analysis by non runtime system. The program analyzer was first proposed by Sofokleous et al. as BPAS (Basic Program Analyzer System) and this framework is built upon the BPAS with the revision in the runtime system. The BPAS consists of number layers which include IOExecutive, the parser, the walker, the static analyzer and the dynamic analyzer. The IOExecutive receives the input and pass it to the parser. The control flow graph is constructed by the walker layer in the tools. The static and dynamic analysis is performed by the runtime and non runtime system in the tool. Automatic Test Case Generation System (ATCGS) uses two algorithms namely Back-Optimistic (BO) and Clouse-UP (CU) algorithm. To evolve the test case a Genetic Algorithm is used by the BO algorithm and all valid test cases are identified. CU algorithm uncovers the test case that are generated by the BO algorithm and stores it in the repository, it uncovers with the help of control flow graph (CFG). The approach is applied and results shows that it has a high hit ratio of code coverage and the future work planned for the use of system to parse other SUT implemented in other language since the tool is restricted to JAVA platform SUT.

4.5 A Partial Test Oracle:

Kim-Park et al. [34] proposed a system to verify the actual output on XML data. The oracle designed is a partial test oracle where the oracle determines the actual output without knowing the correct output. The oracle verifies the correctness by considering two elements 1) Behavior requirement 2) Oracle Constraint. The XML document is the input to the partial test oracle as SUT and output is generated as a XML document. The specification to the oracle is specified by the behavioral requirement which is given by the tester; the oracle will be more precise if the specification is clear in details. Functions and sets are used to represent the requirement comprising the expected behavior of the SUT. The behavior requirement is used to parameterize the constraints and to find the errors present in SUT. A “Fail” is indicated if any one of the oracle constraint is violated. This approach does not demand for the correct output to be specified to the test oracle.

4.6 Test Oracle generation using Jimple code:

Xu et al. [35] suggested a model for generating test oracle by mining to build a decision tree from the inputs generated from the Java bytecode. A heuristic behavior model is designed to represent the test input and expected result for unit under test (UUT). The behavior model is build from the selected small subset of the test input from the test input generated by a search based technique from the Java bytecode. To simplify the java bytecode a 3-address intermediate representation called Jimple is used [40]. Jimple code is then converted to CFG using soot package. The nodes in the CFG are the predicates in the jimple code, figure 5 represents the generation of the test oracle using jimple code. The second approach is to auto generated input and the generation of decision tree, in which two inputs are generated for predicate, one for true and one for false. The intermediate nodes are the condition and the leaves in the decision tree are the triangle type. The third approach is based on the rule-based test input generation where a specific path is chosen from the CFG of the jimple code, rules are defined for the jimple predicate for the expected output. Later the variable dependent tree (VDT) is generated in which the leaves of the predicate tree (PT) are jimple input variables. Then a value is marked for the leaves by increasing or decreasing its value by applying the rule to PT’s based on the root of the PT. A 94.67% of the mutant was killed using mined test oracle.

![Fig. 5: Jimple code for oracle generation](image)

4.7 Automated Framework for Test Oracle based on multi-network:

Shahamiri et al. [36] addressed three challenges of generating test oracle namely output-domain generation, input domain to output domain mapping and a comparator to check the actual output to the expected output. The first challenge output-domain generation is addressed by I/O relationship analysis [37], where the expected
result vector is prepared with the training sample. A reduced data set is produced by considering the I/O relationship within the application, then the reduced set is expanded and the complete training data set is generated. The second challenge is addressed by the use of multi-network oracle based on several ANN’s which gets the input from the previous section [38]. If the SUT is of complex type which generates several results, then the single-network oracle which is composed of one ANN may not be useful so a multi-network oracle is required. The third challenge is addressed by applying the trained network in the testing process. The experimental result shows that it executes 300 test cases and more than 9000 faults are injected into the SUT, such that 97% of the challenges were automated.

5 Statistical based Oracle:
The statistical based oracle verifies a selected set of sample values where the resources are limited and only a small number of inputs will be included in the test, in which the values are equally likely. The data chosen is not based on any precondition for the randomly selected input and a statistical method is used to verify its correctness.

5.1 Statistical Method for Test Oracle:
Mayer and Guderlei [39] designed an approach for generating the test oracle using the statistical method in which some statistical characteristics are verified for the actual test result. In the case of the randomized software where a random testing is performed, the expected results also will be random. A statistical method is used to test the statistical characteristics of the actual test result. A statistical analyzer and a comparator are designed to perform the test. The statistical analyzer is always based on the implementation under test and cannot be generalized. The comparator collects the n test cases from the statistical analyzer and from the test cases the sample mean and the variance are generated. Two approaches for designing the system were discussed and implemented using java and tested for image analysis. The approach is best suited where the characteristics are known and is based on its second order moments namely mean, variance and so on.

5.2 Log File Analyzer:
Andrew et al. [40] uses a coverage measure and artificial intelligence (AI) based algorithm for the test oracle. A log file is given as an input to the oracle to verify and validate the result obtained. The test oracle involved in the process is called Log file analyzer which is specified as a collection of state machine, were the log file analyzer reads the input from the log file one by one and passes it to state machine. A log file line is noted by the machine and is used for state transition; each transition may have additional conditions. If it could not able to read the line then it reports error. The process is tested with two different applications namely TempControl and WesternWapper. Log File Analysis Language (LFAL) is designed for expressing the current state of the SUT. In order to verify the coverage a core algorithm called GEN is used in machines which has infinite number of states and transitions. The space of log file is viewed as a graph, the GEN algorithm performs a depth first search to obtain a coverage log file suite.

5.3 M-model programs:
Manolache and Kourie [41] proposed a technique using M model programs. Primary program P is modeled to mp1-mpm for the same input, the testing process is repeated until all the defect is covered which is indicated by the disagreement of the output. Both the program and the modeled program has the same property and the economic reason the modeled version of the program are designed to cover only the complex condition of the program under test. The M-mp are executed to generate the output for verification and if there is any disagreement in the output the corrective action is taken. The program under test and the modeled program constitute the test oracle. The approach is tested on complex algorithm software which is more cost effective than the other.

5.4 Automated Test Oracle from system model:
Memon et al. [42] developed a test oracle to test the expected behavior of the GUI. GUI has special challenges due to its characteristics than traditional testing methodology for testing the GUI based application [43, 44]. To automate the test oracle for the GUI based system, first, the system need to be modeled so that it is easy to derive the expected behavior. Next the GUI has to be represented in a form suitable for comparison. A formal model is designed from the GUI specification, which is composed of objects. The GUI actions are represented by its precondition and effects. The oracle automatically receives the action for the test case and the expected state from the model GUI. A verifier is used to verify the expected state with the actual state. Figure 6 explains the oracle generation for GUI system. The technique is tested with a GUI application called the Planning Assisted Tester for graphical user interface Systems (PATHS). Since the technique is platform independent, it will work for any GUI application provided the model GUI is well defined. The technique allows for reuse of operators in GUI and also stores it in the library for reuse in different oracle.
To verify the consistency of the test result the test will be put under re-test. The consistency based test oracle has the advantage in evaluating the effect of change made in one version to the other version, and ensures consistency. The source for the test oracle may be collected from the previous version of the SUT, model of the product or an oracle for the same product from different platform. Fault may occur when the oracle is generated from the historic data taken from the SUT.

6.1 Use of Wrapper Class:

Shukla et al. [45] designed a technique to develop a passive test oracle from the application programmer interface (API) for the components of SUT. The component which is to be tested is put under the wrapper for checking the behavior. The wrapper is also a component and provides more input than applying on its own. The actual function of the component to be tested has a similar interface. The wrappers function that is used to test is the improved version of the member function of the component. A comparative result of the experiment conducted on Symbol Table and Forest in generating the active and passive test oracle results in finding fault effectively with a small number of test cases and no special documentation or tool is required.

6.2 GADGET for Complex Programs:

Michael and McGraw [46] designed software to obtain the condition coverage of C/C++ program called Genetic Algorithm Data GEneration Tool (GADGET). GADGET uses genetic algorithm and uses four test generation optimization techniques namely simulated annealing, gradient descent, a standard genetic algorithm, and a differential genetic algorithm. Test data from the C/C++ program is automatically generated by GADGET tool for a complex program of about 2000 lines. The gradient descent algorithm start with measuring the objective function of the new seed, the neighbor input those results in the best objective function then becomes the new seed and the process continues until the result is obtained. In simulated annealing algorithm the input is
randomly selected and if the newly selected input results in improved objective function it becomes the new seed. The genetic algorithm encodes each input as a string of bits, a fitness value is given to each input for the evaluation of the objective function. Secondly the selection of two inputs at random and crossover is performed. Differential genetic algorithm begins by constructing the population and recombination is done by iterating over the input.

6.3 Daily Automated Regression Tester:

Memon and Xie [47] designed a technique for automatically testing and retesting the graphical user interface by developing a daily automated regression tester (DART). The main aim of the DART is to automate the testing process for GUI. DART starts the automation testing process by first identifying the GUI object and their properties, and formulates the possible number of test cases. DART uses a built in test case generator to generate the smoke test case. A subset is chosen from the total test case and selected for the run. The output is compared with the expected output stored and an execution report is generated to verify the test oracle result. The technique is able to detect 60% of the fault for the most applications by covering a large percentage of the code. It is able to find the defect easily for the longer sequence of event than the short sequence of action and the test cost is comparatively low.

7 Model-based Oracle:

7.1 Test Oracle based on Test Condition:

Lin et al. [48] used software design with test condition to generate test oracle. Size, set relation, order relation and element based test condition in data structure are considered to generate the test oracle. This improves the testability of the software, the behaviors are verified using the test condition selected by a preprocessing system. The test conditions are considered for the specification of the SUT. The result shows that the oracle based on the test condition is more effective and feasible than the conventional assertion technique.

7.2 Test Oracle generation using UML:

Briand and Labiche [49] proposed a methodology, Toolbox for Traffic Engineering Methods (TOTEM) for testing the system derived from the specification of the system. Deriving test requirement is at the end of the analysis development stage. The step involved in TOTEM begins with checking the completeness and correctness of the unified modeling language (UML) diagram which specifies the system. Test requirements are derived from the system sequence diagram and all the requirements are merged to form one test requirement. From the test requirement collected the test cases and test oracle are obtained. The source for the test oracle is the post condition of the operation in a sequence, which is defined by the object constraint language (OCL). The post conditions are formalized in a decision table which is used as a test requirement set.

7.3 Model based Test Oracle for Unit Test:

Padgham et al. [50] developed a fault based model for the units of SUT. The system model designed during the detailed design phase is used as the oracle. The different component such as SUT-Design, SUT-Code, Unit Test Harness and Testing Report are used to form a test framework. The SUT design document is used to extract the information for the generation of the test oracle. The process begins with determining the order of the testing of all the units of the agent SUT and for each unit, test harness is introduced via code augmentation. Then the test harness is executed to generate the test case and runs to test the unit.

7.4 Test Oracle using Complete Test Graph:

D’Souza and Gopinathan [51] provided a solution for generating a complete test graph and the construction of online oracle for the specification S and for the purpose P. In construction of the test oracle, first the complete test graph (CTG) is constructed by moving deterministically in the CTG whenever the input/output is provided by the tester. In construction of the CTG two important assumptions are made, one assuming that the given specification S is deterministic and the other assuming that each module has a unique exit. The product of S and P is computed as described in [52]. A label ‘L2A’ (Lead to Accept) is given to the node that if a state in the node is an acceptable node and REACH_IN, REACH_EXIT nodes are defined.

7.5 Test Oracle Generation Using STALE:

Li and Offutt [53] proposed a model based test oracle technique that uses a model specification language to define the system behavior. Concrete tests are generated from the abstract test which includes test input values, expected output and test oracle. Six new test oracle strategies are proposed to verify different parts of the program. UML state machine language is used as a tool to generate the test. Structural test automation language framework (STALE) is used to read the UML and convert it to a graph, with coverage criteria [54]. The abstract tests are generated using STALE with the coverage criteria specified. Structural Test Automation Framework, a transformation tool to transform the abstract test into concrete test [55]. The test oracle is designed considering
how often to check state and the number of internal states checked. Figure 7 shows the STALE tool and the process of generating test oracle form the UML state machine diagram using Junit.

The Oracle Strategies which checks for the program state, that are State invariant, Object Members, Return values, Parameter Members and after satisfying state invariant are

A) Checking object members in this transition only once.
B) Checking the return value only once.
C) Checking object member and return value only once.
D) Checking the parameter and the return value only once.
E) Checking the parameter, object member and the return value only once.
F) Checking the parameter, object member and the return value for all transition only once.

The model is experimented with several algorithms and it is observed that it is slightly more effective than the base oracle strategies (Null Oracle strategies and State Invariant Oracle strategies).

8 Metrics for Test Oracle:

It is necessary in the software projects that the quality and effectiveness of the process used in the development of the product has to be measured. Metric is the quantitative measurement to measure the quality of the process. It is also important to measure the system components that are used to test the SUT. During the testing process if the metric is not followed then the work completed will not have proper information about the quality of the work. The quality of the test oracle generated has to be measured. Work has stated in the software community to identify the characteristic and measuring the quality of the test oracle. A list of characteristics that can be used as a measure to test the oracle is identified. The characteristics are completeness, accuracy, independence, speed, time of execution and usability.

Fig. 7: STALE tool to generate test oracle from UML state machine

Completeness:

Completeness in the information that is used to define and generate oracle. Irrespective of the type of oracle complete information is required. In [9] where the design document is used to generate the test oracle, the information provided as tabular expression should be complete. With incomplete information, the oracle will not be complete and hence the testing process becomes inefficient which in turn will lead to failure.
Accuracy:
Information provided to generate the oracle should be accurate [24]. Mathematical notations are used to form the oracle, if the notation specified or any other specification that is used for the generation of oracle is not accurate then the oracle will not be able to verify the correct behavior of the SUT.

Independence:
The oracle used to test the system should be independent of algorithm, system platform and the operating platform.

Time of execution:
For the successful completion of a project, the time of execution of the test oracle is important. Some of the factors that affects the time of execution of the test oracle are
1) Considering the scope of the project, that is setting up the test bed, generating the test data and test script, etc.
2) Re-visiting the time estimated for the project since the estimation of test oracle may deviate
3) Availability of the resources for the execution of the project might cause delay in the execution. Many such factors have to be considered and measured.

Usability:
The result that is obtained from the process has to be verified and measured in terms of its usage. In GUI based testing partial results obtained may not be useful to conclude.

9 Review strategies:
To review the result of the work on the survey of the automated test oracle strategies, a systematic and structured approach was followed. The aim of the survey is to know classify the research avenues in automated test oracle:
- Research query 1: Contribution of the test oracle methods
- Research query 2: Cost and effectiveness of the test oracle methods
- Research query 3: Accuracy level of the method used for the generation of the test oracle
The following section includes the main details of the review process and its methods.

9.1 Contribution of the oracle methods:
Fig. 8 classifies the primary studies according to the type of oracle methods. Interestingly all the method contributes a considerable percentage of the work carried out in the automated test oracle research. Specification based oracle contribute 29.6% which is been largely used to automate the test oracle process, about 25.9% test oracle is contributed by heuristic test oracle method, consistency based test oracle makes the least contribution with 11.1%, a 14.5% test oracle is based on statistical techniques and the model based test oracle contributes a 18.5%.

Fig. 8: Contribution of test oracle methods
9.2 Cost and effectiveness of the test oracle methods:

This aim of the analysis is to find the cost involved in generation of the test oracle by various methods of test oracle. Taking our repository of the article of different methods in order to classify the effectiveness of test oracle automation methods.

Table 1: provides a comparison between different methods in terms of the cost and the effectiveness of the test oracle.

<table>
<thead>
<tr>
<th>Test oracle strategies</th>
<th>Cost and effectiveness of the test oracle</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specification and Documentation test oracle</td>
<td>• Expensive for large scale software</td>
</tr>
<tr>
<td></td>
<td>• No difficulty when the size of the software comparatively small</td>
</tr>
<tr>
<td></td>
<td>• Less effective for the large scale software</td>
</tr>
<tr>
<td>Heuristic test oracle</td>
<td>• Vary based on the AI methods</td>
</tr>
<tr>
<td></td>
<td>• Increases difficulty when there is an increase in the decision structure</td>
</tr>
<tr>
<td></td>
<td>• Comparatively more effective in generating the test oracle than other methods with respect to cost involved</td>
</tr>
<tr>
<td>Consistency test oracle</td>
<td>• The result of the method is highly effective</td>
</tr>
<tr>
<td></td>
<td>• Comparatively an expensive method than the heuristic model of the test oracle generation</td>
</tr>
<tr>
<td>Statistical test oracle</td>
<td>• Reduced cost</td>
</tr>
<tr>
<td></td>
<td>• Increasing the reliability of the testing process, still not complete</td>
</tr>
<tr>
<td></td>
<td>• But requires various implementation of the required functions</td>
</tr>
<tr>
<td>Model based test oracle</td>
<td>• Additional cost involved in modeling the system</td>
</tr>
<tr>
<td></td>
<td>• Reduces the complexity when clearly modeled</td>
</tr>
<tr>
<td></td>
<td>• Flow of the event is very clear</td>
</tr>
</tbody>
</table>

Table 1 Cost and effectiveness of the test oracle methods

9.3 Accuracy level of the method:

Considering the accuracy of the method used to generate the test oracle, a study was made. Accuracy of the method means how accurate the results of the test oracle are. The percentage of the test oracle generated by various methods are explain in the fig 9, the percentage of the result can be obtained by comparing the test oracle generated with the expected outcomes specified in the requirement of the system. Clearly indicating that heuristic based test oracle achieved a maximum accuracy of 98% [36].

![Accuracy of test oracle](image)

**Fig. 9:** Accuracy of test oracle generated by various methods

10 Conclusions and Future work:

This paper presents a detailed survey on the various test oracle generation strategies classified in terms of covering specification and documentation, heuristics, consistency, statistical and model based test oracle. This paper has necessarily focused on the automated test oracle strategies and an effort has been made to categorize the strategies based on its behavior. Based on the survey performed it is evident that the automation of test oracle is a potential area of research. The scope of the future work includes constructing tools which can further extend the automation of test oracle generation process, identifying and formulate the process for generation of test oracle based on the nature of the system and type of specification of the system. An improved version of the
machine learning algorithm can be used to increase the accuracy of the test oracle. Result has proven that future work should also focus on executable object level test oracle and automation of test oracle for Internet of Things. Identifying the suitable metric to measure the oracle generation process is expected to minimize the cost, time and effort spend in testing the behavior of the software system.

REFERENCES