A Taxonomy for Test Oracles

Douglas Hoffman
Software Quality Methods, LLC.
Phone 408-741-4830
Fax 408-867-4550
doug.hoffman@acm.org

Keywords: Automated Testing, Model of Testing, Software Under Test, Test Oracles, Test Verification, Test Validation

Abstract

Software test automation is often a difficult and complex process. The most familiar aspects of test automation are organizing and running of test cases and capturing and verifying test results. A set of expected results are needed for each test case in order to check the test results. Generation of these expected results is often done using a mechanism called a test oracle. This paper describes classes of oracles for various types of automated software verification and validation. Several relevant characteristics of oracles are included and the advantages and disadvantages for each class covered.

Background

Software testing is a process of providing inputs to software under test (SUT) and evaluating the results. In software testing, the mechanism used to generate expected results is called an oracle. (In this paper, the first letter will be capitalized when referring to an Oracle for a specific test.) Many different approaches can be used to generate, capture, and compare test results. The author, for example, at one time or another has used the following methods for generating expected results:

- Manual verification of results (human oracle)
- Separate program implementing the same algorithm
- Simulator of the software system to produce parallel results
- Debugged hardware simulator to emulate hardware and software operations
- Earlier version of the software
- Same version of software on a different hardware platform
- Check of specific values for known responses
- Verification of consistency of generated values and end points
- Sampling of values against independently generated expected results

Test automation usually requires incorporation of Oracles into the testing process so test outcomes can be evaluated. Automating the verification of results has significant implications on both the test case and Oracle design. Because of the current high machine speeds and low cost of memory, test cases can generate very large amounts of data, with corresponding amounts of Oracle data needed for comparison. One or both sets of data can be generated and stored for comparison and then discarded if no differences are found. When data comparison is incorporated into test cases, effort is required to design each test to include error handling, reporting differences, and capturing error results. When the comparisons are done separately the effort is not repeated, but standards must be employed for formatting and storing inputs and results.

Many organizations today depend on a human oracle to verify test results. The tester is expected to know how the software will work, and they are expected to know when the software misbehaves. This often happens by default for manual testing, and is usually the case for GUI testing. A human oracle is not satisfactory for several reasons when test cases are automated. The volume of data from automated tests is often overwhelming. A person may not keep up with analyzing displayed information before the system changes it. Not all effects of a test case are available and displayed for a person to observe. The automated testing process is tedious and requires concentration for arbitrarily long periods. A person also becomes quickly trained on what to expect, and once trained is likely to overlook minor deviations (errors).

A worse situation occurs with automated tests when tests run without benefit of any verification. The result from merely running a test is nearly always the same whether or not a fault is encountered – program termination. Based on experience, very few errors cause noticeable abnormal test termination. Unless test results are verified it requires a spectacular event to show that an error has occurred. When a batch of automated tests is run with only cursory checks, we may only learn that something went wrong somewhere, without a clue about the likely cause. Some automated mechanism is needed to check the results from automated tests.

Creating an oracle to verify values for a mathematical subroutine may be straightforward by using a different algorithm, language, compiler, etc. At the other extreme, an Oracle for the interrupt handling of an operating system kernel is far more difficult to create. Hardware and system emulators need to be created, and parallel mechanisms for causing specific events need to be put in place for both the SUT and the Oracle. Timing and synchronization between the SUT and Oracle are also extremely difficult to manage to correctly verify software operation.

The difficulty in creating most test oracles falls somewhere between the two extremes. It is often impractical to generate complete sets of expected results. It is particularly difficult to generate expected information for file directories, machine registers, system tables, memory, etc. Usually these aspects and side-effects of the SUT are ignored when tests are verified unless there is a gross, obvious problem. This is also true when the tests are manually run.

A Simple Model for Automated Tests

Figure 1 shows an Input-Process-Output model for black box testing. The test case is a set of inputs and verification is done by observing the results. SUT’s very seldom fit this model, however, as they have multiple, complex inputs and results. We need to know the values for all of the inputs and check all of the results in order to know whether the SUT responds properly. Also, some of the results from software execution are only indirectly related to the functions we are exercising in our test. Test results include such things as residual values left in memory, program states for the SUT and other software, instrument control signals, and data base values.

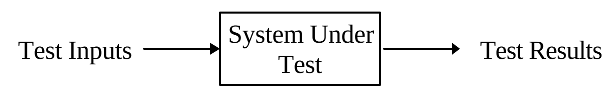

Figure 1: I-P-O Testing Model

Figure 2 shows a more complete model for software testing, including more categories of inputs to and results from a test. To determine whether the SUT responds properly, we need to know or set all of the inputs and check all of the results. Because of the vast possible outcomes from running a program, test designers select what they consider are relevant inputs and results, and then choose a subset of these to use in predicting and verifying program behavior. The test case input values are only one part of the stimulus for a test, and even thorough test plans identify only some of the test case preconditions. The environmental inputs are seldom spelled out in detail.

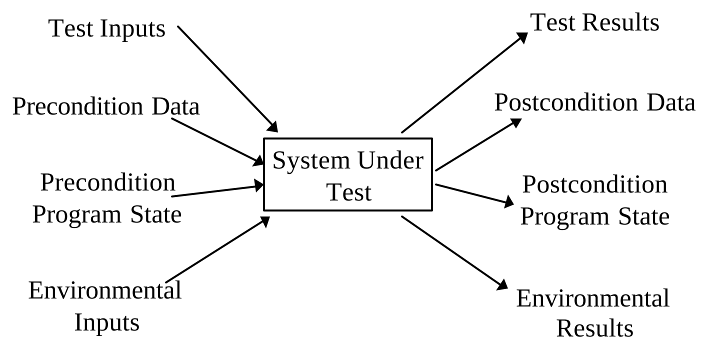

Figure 2: Expanded Testing Model

Several observations can be made when introducing an oracle into the model. Different types of oracles are needed for different types of software. The domain, range, and form of input and output data varies substantially between programs. Most software has multiple forms of inputs and results so several oracles may be needed for a single software program. Different characteristics in a program may require separate oracles. For example, a program's results may include computed functions, screen navigations, and asynchronous event handling. Several oracles may need to work together because of interactions of common inputs. In the case of a word processor, pagination changes are based upon characteristics such as the font and font size, while the test case may be about color compatibility. An oracle for pagination has to factor in fonts even when a test case is about color. Although an oracle may be excellent at predicting certain results, only the SUT running in the target environment will process all of the inputs and provide all of the results. No matter how meticulous we are in creating an oracle, we will not achieve both independence and completeness.

Because using a single oracle may be impractical to model all system behaviors for the SUT, this paper will assume that oracles are created for specific purposes. This simplifying assumption holds since an oracle that completely models SUT behavior can be considered to be composed of several special purpose oracles focusing on specific SUT behaviors. The special purpose oracle can then completely predict SUT behaviors for which it is designed. We can add other oracles to predict other behaviors and results from the SUT. (In practice, most test oracles focus on modeling straightforward behaviors, and we apply different oracles at different times to check program behaviors such as functionality, screen navigations, or memory use.) The characteristics of these focused oracles can be at the extremes of our measurements.

Characteristics of Oracles

There are several characteristics we might measure relating an oracle to the SUT. **Table 1** provides a list of some useful measures for oracles. Each of these characteristics describe a correspondence between an oracle and the SUT and measures can range from no relationship to exact duplication. Completeness, for example, can range from no predictions (which is not very useful) to exact duplication in all results categories (a second implementation of the SUT).

- Completeness of information from oracle
- Accuracy of information from oracle
- Independence of oracle from SUT
    - Algorithms
    - Sub-programs and libraries
    - System platform
    - Operating environment
- Speed of predictions
- Time of execution of oracle
- Usability of results
- Correspondence (currency) of oracle through changes in the SUT

Table 1: Oracle Characteristics

It is easy to see that the more complete and accurate an oracle is, the more complex it has to be. Indeed, if the oracle exactly predicts all results from the SUT it will be at least as complex. This also means that the better an oracle is at providing expected results, the more likely that detected differences are due to faults in the oracle rather than the SUT. Likewise, the more an oracle predicts about program state and environment conditions, the more dependent the oracle is on the SUT and operating environment. This dependence makes the oracle more complex and more difficult to maintain. It also means that faults may be missed because both the SUT and the oracle may contain the fault.

Software tests themselves can be classified in many different ways. Manual testing brings up images of a human providing input and interpreting results as the means of testing. Yet, humans sometimes need books, tables, calculators, or even programs (an Oracle) to know the expected result. Automated testing does not mean mechanical reproduction of manual tests. Automated tests that include evaluation of results need some kind of oracle regardless of the type or purpose of the tests. Yet, the mechanism for evaluation of results ranges from none (the program or system didn't crash) to exact (all values, displays, files, etc., are verified). Various levels of effort and exactness are appropriate under different circumstances. The nature and complexity of an oracle is also dependent upon those circumstances.

Types of Oracles

Real world oracles vary widely in their characteristics. Although the mechanics of various oracles may be vastly different, a few classes can be identified which correspond with automated test approaches. These types of oracles are categorized based upon the outputs from the oracle rather than the method of generation of the results. Thus, an oracle that uses a lookup table to derive values may be the same type of oracle as one that implements an alternate algorithm to compute the values. The type descriptions define the purpose of the oracle and its method of use. Five types are identified and defined below. They are labeled True, Stochastic, Heuristic, Sampling, and Consistent oracles.

A "True oracle" faithfully reproduces all relevant results for a SUT using independent platform, algorithms, processes, compilers, code, etc. The same values are fed to the SUT and the Oracle for results comparison. The Oracle for an algorithm or subroutine can be straightforward enough for this type of oracle to be considered. The *sin()* function, for example, can be implemented separately using different algorithms and the results compared to exhaustively test the results (assuming the availability of sufficient machine cycles). For a given test case all values input to the SUT are verified to be "correct" using the Oracle's separate algorithm. The less the SUT has in common with the Oracle, the more confidence in the correctness of the results (since common hardware, compilers, operating systems, algorithms, etc., may inject errors that effect both the SUT and Oracle the same way). Test cases employing a true oracle are usually limited by available machine time and system resources.

A "Stochastic" approach focuses on verifying a statistically selected sample of values. This is most useful when resources are limited and only a relatively small amount of inputs will be included in the tests. For all inputs and ranges for the inputs, values are selected which are equally likely. For the *sin()* example, a pseudo-random number generator may be used to select the input values. The same values are fed to the SUT and the Oracle for results comparison. The statistically random input selection results in a test case that has no bias from the data chosen. It also means that suspect or error prone areas of the software are no more or less likely to be encountered than any other area. Either the Oracle has to be substantial enough to be able to accept arbitrary inputs or the pseudo-random sequence needs to be known in advance and an Oracle created for those particular values.

A "Heuristic oracle" reproduces selected results for the SUT and the remaining values can be checked using simpler algorithms or consistency checks based on a heuristic. For the *sin()* function, a Heuristic Oracle might generate only the specific values for $sin(\pi/2)$, $sin(\pi)$, $sin(3\pi/2)$, $sin(2\pi)$ [whose results are 1, 0, -1, 0]. The test can then give values between the four points at very small increments to the SUT. A heuristic is applied to verify that the SUT returns values that are progressively greater (or less) than the last value. Although the heuristic approach will accept many functions that are incorrect, the Oracle is very easy to implement (especially when compared to a True Oracle), runs much faster, and will find most faults.

The "Sampling" approach uses a selected set of values. The values are selected because of some criteria other than statistical randomness. Boundary values, specific integers, midpoints, minima, and maxima are examples often chosen when testing. Often, values are selected because they are easy to generate, recognize, or recall. (These are all selected samples that are not statistically random.) Once the values are selected, an Oracle can be created that provides the expected reslults. Software testing usually includes some effort based on Sampling to focus on areas likely to have faults and critical functions and features. The key difference between the Stochastic oracle and Sampling oracle is in the method of selection of input and result values.

A "Consistent"oracle uses the results from one test run as the Oracle for subsequent tests. This is particularly useful for evaluating the effects of changes from one revision to another. The Oracle in this situation comes from a simulator, equivalent product, software from an alternate platform, or an early version of the SUT. The values being compared can include intermediate results, call trees, data values, or any other data extracted from the SUT automatically. The Oracle-generated data is usually too voluminous to be thoroughly or exhaustively verified. The value in comparing results from the SUT and the Oracle is from evaluating and explaining any differences. Because very large volumes of data can be stored and compared, the test cases can cover large input and result ranges. Although historic faults may remain when this technique is used, new faults and side-effects are often exposed and fixes are confirmed.

Table 2 summarizes the five types of oracles and some of their characteristics.

	True Oracle	Stochastic	Heuristic	Sampling	Consistent
Definition	Independent generation of expected results	Verify a randomly selected sample	Verify selected points, use a heuristic for remainder	Verify a specially selected sample	Compare run *n* results with *n-1*
Example of use	Algorithm Validation	Operational Verification	Algorithm Verification	Boundary Testing	Regression Test
Advantages	Possibility for exhaustive testing	Can automate tests with a simple Oracle	Easier than True Oracle	Very fast verification possible with simple Oracle	Fastest; Can generate and verify large amounts of data
Dis-advantages	Expensive implementation. Possibly long execution times	May miss systematic and specific errors. Can be time consuming to verify	Can miss systematic errors and incorrect algorithms	May Miss Systematic or Specific Errors	Original run may include unknown errors

Table 2: Five Types of Oracles

Other Remarks on Oracles

Data from the Oracle can be generated before, parallel to, or after the test case is run. If the Oracle data is generated before the test, the inputs for the test case need to be known and the expected results must be stored in suitable form for comparison during or after testing. Early Oracle data generation is useful when the Oracle is slow, and it is required for the consistency approach. When the test case performs comparisons with expected results the Oracle has to run before or in parallel with the test case. Parallel running of an Oracle presumes that the Oracle runs quickly enough to be practical. When test results are stored and checked after test execution, the timing of Oracle data generation can be independent of test execution. Such after-the-test verification can be done using stored results from a test run with either stored or real-time generated Oracle output.

Test results can be verified manually, within the test case, or automated separately. Manual verification requires both test results and Oracle data be available for comparison and is limited by human processing capabilities. Verification within a test case means that the Oracle data has to be available when the test case runs, which means either prior or parallel running of the Oracle. The test case also needs to be designed to perform the collection, comparing, and reporting of results. Separate automation of results comparison requires that results from the test run are saved and that either the Oracle results are likewise saved or generated as needed by the verification routines.

Care must be taken during test planning to decide on the method of results comparison. Oracles are required for verification and the nature of an oracle depends on several factors under the control of the test designer and automation architect. Different Oracles may be used for a single automated test and a single oracle may serve many test cases. If test results are to be analyzed, some type of oracle is required.

Douglas Hoffman
Software Quality Methods, LLC.
Phone 408-741-4830
Fax 408-867-4550
doug.hoffman@acm.org

Bio:

Douglas Hoffman is an independent consultant with Software Quality Methods, LLC. He has been in the software engineering and quality assurance fields for over 25 years and now teaches courses and consults with management in strategic and tactical planning for software quality. For five years he served as Chairman of the Santa Clara Valley Software Quality Association (SSQA), a Task Group of the American Society for Quality (ASQ). He has been a participant at dozens of software quality conferences and has been Program Chairman for several international conferences on software quality. He is a member of the ACM and IEEE and is active in the ASQ as a Senior Member, participating in the Software Division, the Santa Clara Valley Section, and the Software Quality Task Group. He is Certified by ASQ as a Software Quality Engineer and has been a registered ISO 9000 Lead Auditor. He has a BA in Computer Science, an MS in Electrical Engineering, and an MBA.

Douglas' experience includes consulting, teaching, managing, and engineering in the computer systems and software industries. He has over fifteen years experience in creating and transforming software quality and development groups, and twenty years of business management experience. His work in corporate, quality assurance, development, manufacturing, and support organizations makes him very well versed in technical and managerial issues in the computer industry. Douglas has taught technical and managerial courses in high schools, universities, and corporations for over 25 years.