Wikipedia
The Free Encyclopedia

Test oracle

In computing, software engineering, and software testing, a **test oracle** (or just **oracle**) is a mechanism for determining whether a test has passed or failed.[1] The use of oracles involves comparing the output(s) of the system under test, for a given test-case input, to the output(s) that the oracle determines that product should have. The term "test oracle" was first introduced in a paper by William E. Howden.[2] Additional work on different kinds of oracles was explored by Elaine Weyuker.[3]

Oracles often operate separately from the system under test.[4] However, method postconditions are part of the system under test, as automated oracles in design by contract models.[5] Determining the correct output for a given input (and a set of program or system states) is known as the **oracle problem** or test oracle problem,[6]:507 which is a much harder problem than it seems, and involves working with problems related to controllability and observability.[7]

Categories

A research literature survey covering 1978 to 2012[6] found several potential categories of test oracles.

Specified

These oracles are typically associated with formalized approaches to software modeling and software code construction. They are connected to formal specification,[8] model-based design which may be used to generate test oracles,[9] state transition specification for which oracles can be derived to aid model-based testing[10] and protocol conformance testing,[11] and design by contract for which the equivalent test oracle is an assertion.

Specified Test Oracles have a number of challenges. Formal specification relies on abstraction, which in turn may naturally have an element of imprecision as all models cannot capture all behavior.[6]:514

Derived

A derived test oracle differentiates correct and incorrect behavior by using information derived from artifacts of the system. These may include documentation, system execution results and characteristics of versions of the system under test.[6]:514 Regression test suites (or reports) are an example of a derived test oracle - they are built on the assumption that the result from a previous system version can be used as aid (oracle) for a future system version. Previously measured performance characteristics may be used as an oracle for future system versions, for example, to trigger a question about observed potential performance degradation. Textual documentation from previous system versions may be used as a basis to guide expectations in future system versions.

A pseudo-oracle[6]:515 falls into the category of derived test oracle. A pseudo-oracle, as defined by Weyuker,[12] is a separately written program which can take the same input as the program or system under test so that their outputs may be compared to understand if there might be a problem to investigate.

A partial oracle[6]:515 is a hybrid between specified test oracle and derived test oracle. It specifies important (but not complete) properties of the system under test. For example, metamorphic testing exploits such properties, called metamorphic relations, across multiple executions of the system.

Implicit

An implicit test oracle relies on implied information and assumptions.[6]:518 For example, there may be some implied conclusion from a program crash, i.e. unwanted behavior - an oracle to determine that there may be a problem. There are a number of ways to search and test for unwanted behavior, whether some call it negative testing, where there are specialized subsets such as fuzzing.

There are limitations in implicit test oracles - as they rely on implied conclusions and assumptions. For example, a program or process crash may not be a priority issue if the system is a fault-tolerant system and so operating under a form of self-healing/self-management. Implicit test oracles may be susceptible to false positives due to environment dependencies. Property based testing relies on implicit oracles.

Human

When specified, derived or implicit test oracles cannot be used, then human input to determine the test oracles is required.[7] These can be thought of as quantitative and qualitative approaches.[6]:519–520 A quantitative approach aims to find the right amount of information to gather on a system under test (e.g., test results) for a stakeholder to be able to make decisions on fit-for-purpose or the release of the software. A qualitative approach aims to find the representativeness and suitability of the input test data and context of the output from the system under test. An example is using realistic and representative test data and making sense of the results (if they are realistic). These can be guided by heuristic approaches, such as gut instincts, rules of thumb, checklist aids, and experience to help tailor the specific combination selected for the program/system under test.

Examples

Test oracles are most commonly based on specifications and documentation.[13][14] A formal specification used as input to model-based design and model-based testing would be an example of a *specified test oracle*. The *model-based oracle* uses the same model to generate and verify system behavior.[15] Documentation that is not a full specification of the product, such as a usage or installation guide, or a record of performance characteristics or minimum machine requirements for the software, would typically be a derived test oracle.

A consistency oracle compares the results of one test execution to another for similarity.[16] This is another example of a derived test oracle.

An oracle for a software program might be a second program that uses a different algorithm to evaluate the same mathematical expression as the product under test. This is an example of a pseudo-oracle, which is a derived test oracle.[12]:466

During Google search, we do not have a complete oracle to verify whether the number of returned results is correct. We may define a metamorphic relation[17] such that a follow-up narrowed-down search will produce fewer results. This is an example of a partial oracle, which is a hybrid between specified test oracle and derived test oracle.

A statistical oracle uses probabilistic characteristics,[18] for example with image analysis where a range of certainty and uncertainty is defined for the test oracle to pronounce a match or otherwise. This would be an example of a quantitative approach in human test oracle.

A heuristic oracle provides representative or approximate results over a class of test inputs.[19] This would be an example of a qualitative approach in human test oracle.

References

1. Kaner, Cem; *A Course in Black Box Software Testing* (http://www.testingeducation.org/k04/OracleExamples.htm), 2004
2. Howden, W.E. (July 1978). "Theoretical and Empirical Studies of Program Testing". *IEEE Transactions on Software Engineering*. **4** (4): 293–298. doi:10.1109/TSE.1978.231514 (https://doi.org/10.1109%2FTSE.1978.231514).
3. Weyuker, Elaine J.; "The Oracle Assumption of Program Testing", in *Proceedings of the 13th International Conference on System Sciences (ICSS), Honolulu, HI, January 1980*, pp. 44-49
4. Jalote, Pankaj; *An Integrated Approach to Software Engineering*, Springer/Birkhäuser, 2005, ISBN 0-387-20881-X
5. Meyer, Bertrand; Fiva, Arno; Ciupa, Ilinca; Leitner, Andreas; Wei, Yi; Stapf, Emmanuel (September 2009). "Programs That Test Themselves". *Computer*. **42** (9): 46–55. doi:10.1109/MC.2009.296 (https://doi.org/10.1109%2FMC.2009.296).
6. Barr, Earl T.; Harman, Mark; McMinn, Phil; Shahbaz, Muzammil; Yoo, Shin (November 2014). "The Oracle Problem in Software Testing: A Survey" (http://discovery.ucl.ac.uk/1471263/1/06963470.pdf) (PDF). *IEEE Transactions on Software Engineering*. **41** (5): 507–525. doi:10.1109/TSE.2014.2372785 (https://doi.org/10.1109%2FTSE.2014.2372785).
7. Ammann, Paul; and Offutt, Jeff; "Introduction to Software Testing", *Cambridge University Press*, 2008, ISBN 978-0-521-88038-1
8. Börger, E (1999). Hutter, D; Stephan, W; Traverso, P; Ullman, M (eds.). *High Level System Design and Analysis Using Abstract State Machines*. *Applied Formal Methods — FM-Trends 98*. Lecture Notes in Computer Science. Vol. 1641. pp. 1–43. CiteSeerX 10.1.1.470.3653 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.470.3653). doi:10.1007/3-540-48257-1_1 (https://doi.org/10.1007%2F3-540-48257-1_1). ISBN 978-3-540-66462-8.
9. Peters, D.K. (March 1998). "Using test oracles generated from program documentation". *IEEE Transactions on Software Engineering*. **24** (3): 161–173. CiteSeerX 10.1.1.39.2890 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.39.2890). doi:10.1109/32.667877 (https://doi.org/10.1109%2F32.667877).
10. Utting, Mark; Pretschner, Alexander; Legeard, Bruno (2012). "A taxonomy of model-based testing approaches" (https://eprints.qut.edu.au/57853/1/master_pdflatex.pdf) (PDF). *Software Testing, Verification and Reliability*. **22** (5): 297–312. doi:10.1002/stvr.456 (https://doi.org/10.1002%2Fstvr.456). ISSN 1099-1689 (https://www.worldcat.org/issn/1099-1689).
11. Gaudel, Marie-Claude (2001). Craeynest, D.; Strohmeier, A (eds.). *Testing from Formal Specifications, a Generic Approach*. *Reliable SoftwareTechnologies — Ada-Europe 2001*. Lecture Notes in Computer Science. Vol. 2043. pp. 35–48. doi:10.1007/3-540-45136-6_3 (https://doi.org/10.1007%2F3-540-45136-6_3). ISBN 978-3-540-42123-8.
12. Weyuker, E.J. (November 1982). "On Testing Non-Testable Programs" (https://doi.org/10.1093%2Fcomjnl%2F25.4.465). *The Computer Journal*. **25** (4): 465–470. doi:10.1093/comjnl/25.4.465 (https://doi.org/10.1093%2Fcomjnl%2F25.4.465).
13. Peters, Dennis K. (1995). *Generating a Test Oracle from Program Documentation* (M. Eng. thesis). McMaster University. CiteSeerX 10.1.1.69.4331 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.69.4331).

14. Peters, Dennis K.; Parnas, David L. "Generating a Test Oracle from Program Documentation" (http://www.engr.mun.ca/~dpeters/papers/issta.pdf) (PDF). *Proceedings of the 1994 International Symposium on Software Testing and Analysis*. ISSTA. ACM Press. pp. 58–65.
15. Robinson, Harry; *Finite State Model-Based Testing on a Shoestring* (http://www.harryrobinson.net/MBT-on-a-shoestring.pdf), STAR West 1999
16. Hoffman, Douglas; *Analysis of a Taxonomy for Test Oracles* (http://www.softwarequalitymethods.com/Papers/OracleTax.pdf), Quality Week, 1998
17. Zhou, Z.Q.; Zhang, S.; Hagenbuchner, M.; Tse, T.H.; Kuo, F.-C.; Chen, T.Y. (2012). "Automated functional testing of online search services". *Software Testing, Verification and Reliability*. **22** (4): 221–243. doi:10.1002/stvr.437 (https://doi.org/10.1002%2Fstvr.437). hdl:10722/123864 (https://hdl.handle.net/10722%2F123864).
18. Mayer, Johannes; Guderlei, Ralph (2004). "Test Oracles Using Statistical Methods" (http://www.mathematik.uni-ulm.de/sai/mayer/publications/oracles.pdf) (PDF). *Proceedings of the First International Workshop on Software Quality, Lecture Notes in Informatics*. First International Workshop on Software Quality. Springer. pp. 179–189.
19. Hoffman, Douglas; Heuristic Test Oracles (http://www.softwarequalitymethods.com/Papers/STQE%20Heuristic.pdf), Software Testing & Quality Engineering Magazine, 1999

Bibliography

- Binder, Robert V. (1999). "Chapter 18 - Oracles" in *Testing Object-Oriented Systems: Models, Patterns, and Tools*, Addison-Wesley Professional, 7 November 1999, ISBN 978-0-201-80938-1