A Systematic Investigation of Software Test Oracle Metrics: Current Trends, Future Directions

Manas Kumar Yogi[1] K. Chandrasekhar[2]
[1,2]Assistant Professor
[1,2]Department of Computer Science & Engineering
[1,2]Pragati Engineering College (Autonomous), Surampalem, Kakinada, A.P. India

Abstract— Test oracles are essential to verify the testing procedure. Here the role of test metrics for making an strong, efficient test oracle is indispensible. This paper presents the principles of construction of test oracles using pseudo oracles and metamorphic relations. We have also discussed how future work in this area can be carried out using mechanisms of extended regression testing and extended metamorphic testing. Our paper acts as a readymade guide for software testing practitioners who want to know the way to work with test oracles and be cognizant about the issues which pose difficulty to work with test oracles.

Key words: Oracle, Regression, Metamorphic, Automation, Functionality

I. INTRODUCTION

Distinguishing between the correct and incorrect behavior of the software under test can be performed by specialized software called the software test oracle. The need of automated test oracles to save testing time is obvious given complex applications used in modern era but still it is an open problem to automate test oracles given the uncertainty in test specifications. Where there is no full specification of the properties of the system under test, one may hope that it is possible to construct some form of partial oracle that is able to answer oracle questions for some inputs, relying on alternative means to answer the oracle question for others. Such partial oracles can be constructed using metamorphic testing (based on known relationships between desired behaviour) or by deriving oracle information from executions or documentation. Its found that for many systems and much of testing as currently practiced in industry, the tester does not have luxury of formal specification nor assertions, nor even automated partial oracles. The tester must therefore face the potentially complex task of manually checking the system's functionality for all test cases generated. In such cases, it is essential that automated software testing approaches address the human oracle cost problem.

II. CURRENT TRENDS IN SOFTWARE TEST ORACLE DEVELOPMENT

We need various metrics to develop a robust test oracle. Metrics are essential to control, improve the quality of test oracles. The test oracle metrics helps in taking decision for next phase of activities, evidence of the prediction, taking decision on process change, understanding the type of improvement needed.

A. Usage of Pseudo Test Oracles

The so called non testable programs used pseudo test oracles. Formally it is defined as a test oracle D which accepts test activity sequences of the form:

$$f_1(x)O_1\ f_2(x)\ O_2{:}[\ f_1 \neq f_2 \wedge O_1 = O_2] \quad (1)$$

Where f_1, f_2 are alternative versions of the components of the software under test on the same value.

From the above equation we can state that the metrics involved in construction of pseudo test oracles are number of comments, help documents size, count of the requirements specifications. As without these metrics construction of f_1 and f_2 are not possible. Subsequently to make a robust test oracle the textual documentation is very much useful.

B. Application of metamorphic relations for generating test oracles

In this technique, properties of the software under test are checked after partial test execution. Metamorphic relations are inferred manually after white box inspection of the software under test. a metamorphic relation relates different executions, not necessarily on the same input, of the same implementation relative to its specification. A metamorphic relation is shown as below:

$$f(x_1)O_1 f(x_2)O_2 \ldots f(i_k)O_k : [expr \wedge\ k >= 2] \quad (2)$$

Where expr is a constraint, usually arithmetic, over the inputs x_i and O_i. This definition makes clear that a metamorphic relation is a constraint on the values of stimulating the single SUT for at least twice, observing the responses, and imposing a constraint on how they interrelate. When the SUT is nondeterministic, such as a classifier whose exact output varies from run to run, defining metamorphic relations solely in terms of output equality is usually insufficient during metamorphic testing. We can clearly observe that the metrics used in developing metamorphic test oracle includes the requirements specification document. The more clear the needs are stated in this document the more efficient the construction of the test oracle will be.

III. FUTURE WORK IN TEST ORACLE DEVELOPMENT

A. Extended Metamorphic Testing

Metamorphic testing has great strength as a medium to overcome the lack of an oracle and it continues to be extended for instance in fields of context-sensitive middleware and service oriented software. In this section, we consider other possible extensions to metamorphic testing. We have already observed that metamorphic testing can be treated as a special case of property testing for an arbitrary property p that must respect the oracle D, for all test activity sequences, σ so Dσ if pσ. This simple

All rights reserved by *www.ijsrd.com*

generalisation reveals that metamorphic testing is a form of property testing and, thereby, allows us to consider other forms of property testing that retain the essence of metamorphic testing, but extend current metamorphic concepts. For example, we can, rather trivially, extend the sequence of input-output pairs. We need not consider the metamorphic property to be a 4-ary predicate, but can simply regard it as a unary predicate on arbitrary length test activity sequences, a simple generalisation that allows metamorphic testing of the form:

$$D(x_1, y_1, \ldots, x_k, y_k, R, x_{k+1}, y_{k+1}, \ldots, x_n, y_{ni}) \text{ if } \pi(x_1, y_1, \ldots, x_n, y_n)$$

For some sequence of n stimulus-observation pairs, the first k of which occur before the reset and the final n−k of which occur after the reset. We can also relax the constraint that the test input should contain only a single reliable reset to allow multiple resets as is common in state-based testing. We could also relax the constraint that between the reliable resets where there is exactly one stimulus and one observation. Perhaps metamorphic testing would be more useful if it allowed a more complex interplay between stimuli and observations. The constraint that the oracle would place on such a relaxed notion of metamorphic testing would be $D\sigma$ if $\pi\sigma$ for all test activity sequences σ that contain at least one reliable reset. This raises the question about whether or not we should even require the presence of a single reliable reset operation,in case, were we to relax this constraint and allow arbitrarily many reliable resets (including none) then we would have generalized Metamorphic Testing to the most general case; simply property testing, formulated within our framework in terms of test activity sequences.
That is

$$D\sigma \text{ if } \pi\sigma$$

This looks like an over-generalization as it does not seem to retain the spirit of metamorphic testing. For instance, using such a formulation, we could capture the property that the first n outputs (observations) of the system under test should be sorted in ascending order and should appear without any input being required (no stimuli):

$$D(y_1, \ldots, y_{ni}) \text{ if } \forall\, i.1 \leq i < n \cdot yi \leq yi+1$$

This does not have any similarity with the original definition of metamorphic testing because there is no concept of relating one input-output observation to another. It seems that the concept of the reliable reset is central to the notion of metamorphic testing. Perhaps a generalised definition of metamorphic testing that may prove useful for future work would define a metamorphic relation to be a relation on a sequence of at least two test sequences, separated by reliable resets and for which each test sequence contains at least one observation preceded by at least one stimulus. Recall that a test sequence contains at least one observation activity preceded by at least one stimulus activity, and that stimuli and observations, are deliberately very general: the stimulus could merely consist of calling the main function, starting the system or resetting, while the observation could be that the system terminates within a specified time without any output.

This makes clear the effect of reliable resets and the limitation that a metamorphic test process involves at least two test sequences. Finally, having formally defined a (general) metamorphic test sequence, we can define Generalised Metamorphic Testing (GMT) as follows: Given a metamorphic test sequence, Σ, GMT respects the oracle constraint $D\Sigma$ if $\pi\Sigma$.

That is, GMT is nothing more than property testing on (general) metamorphic test sequences. Using GMT we can capture more complex metamorphic relationships than with traditional metamorphic testing. For example, suppose that a tester has three stimuli activities available, each corresponding to different channels. Stimulus s1 stimulates an output on channel c1, while stimulus s2 stimulates an output on channel c2. There is a third channel that must always contain a log of all outputs on the other two channels. The contents of this third channel are output in response to stimulus s3. This metamorphic logging relationship, πL, can be captured quite naturally by GMT:

$$\pi_L(s_1, y_1, R, s_2, y_2, R, s_3, y_3) \text{ iff } \{y_1, y_2\} \subseteq y_3.$$

We can also capture properties between multiple executions that rely on persistence, the archetype of which is a persistent counter. For example, the generalised metamorphic relation below, πP , captures this archetype of persistence that stimulus t causes the observation c, which reports a count of the number of previous stimuli s that have occurred. This analysis shows that there may be interesting and potentially useful generalisations of metamorphic testing. Future work on metamorphic oracles could explore the practical implications for these more general forms of metamorphic relation, such as those we briefly discussed above.

B. Extending Regression Testing

Software Product Lines (SPLs) are sets of related versions of a system. A product line can be represented as a tree of related software products in which branches store new alternative versions of the system, each of which shares some core functionality contained by a base version. As SPLs are about versions of a system, there should be a connection between regression testing and SPL testing upon which the definition of oracles may hold true. We should be able to formulate oracles for SPLs in a similar way to those for regression testing. The sequence of releases to which we might apply regression testing have to be merely a special case of an SPL testing, in which the degree of the tree is 1 and thus the tree is a degenerate tree (i.e., a sequence). Recall that, for regression testing, we have two releases, R1 and R2 for which we have oracles D1 and D2 respectively. We clarified the distinction between perfective maintenance, corrective maintenance and changed functionality. In corrective maintenance, the implementation is found to be faulty and has to be fixed, but this has no impact on the oracle, which remains unchanged: D1 = D2.

In perfective maintenance R2 adds new functionality to R1 (in which case the domains of D1 and D2 are expected to be distinct): $dom(D1) \cap dom(D2) = \emptyset$ With changed functionality, the requirements are altered so that the new implementation must change the behaviour of the previous implementation, not because the previous version is incorrect (which would have been merely an instance of corrective maintenance), but because the requirements have changed. In this 'changed requirements' case, there is a clash between the two oracles:

$$\exists\sigma \cdot D1\sigma \neq D2\sigma$$

All rights reserved by www.ijsrd.com

Suppose we have two branches of an SPL, B1 and B2, each of which shares the functionality of a common base system B. We can think of the sequence of releases: B followed by B1 as a regression testing sequence. Similarly, we can think of B followed by B2 as a different regression test sequence. Now suppose that the oracles for B, B1 and B2 are D, D1 and D2 respectively. If D = D1 then this branch of the SPL is really a correction to the base system and it should be merged back into the base system (and similarly for the case where D = D2). Subsequently, we can see that if D = D1 = D2 then there is no need for a branch at all; the same oracle, D can be used to test all versions of the system, and so we can assert that there are no separate versions. It is not hard to guess how such a context might arise in practice: an engineer, burdened with many bug fix requests, is concerned that some specific big fix is really a feature extension and so a new branch is created for a specific customer to implement their 'bug fix' request, while leaving the existing base system as a separate line in the SPL (for all other customers). This is understandable, but it is not good practice. By reasoning about the oracles and their relationships, we can determine whether such a new branch is truly required at the point at which it might be created. This observation leads us to define the minimal requirements on the oracle of a version for it to be necessary to create a new SPL branch. Suppose we branch into two new versions. This is a typical (but special) case, that easily generalises to the case of one and n new branches. We expect that the two new versions extend the base system with new functionality and that they do so without interfering with the existing core functionality of the base system and that each extension is conflicting, so that two branches are required, one for each different version. More formally:

$$dom(D) \cap dom(D1) = \emptyset$$
$$dom(D) \cap dom(D2) = \emptyset$$
$$\exists \sigma \cdot D1\sigma \; 6= \; D2\sigma$$

We can see that this is none other than requiring that the two branches are 'perfective maintenance' activities on the base system and that each has changed requirements. Where these constraints are not met, then the branches may be unnecessary and could be combined. Using such a simple formulation we might search for such "mergeable" branches.

C. Generalization of Oracles

We predict that the future work may also consider the way in which our notion of oracles could be generalized and the theoretical properties of oracles. This section explores more theoretical possibilities for future work, but exploring probabilistic notions of oracles and concepts of soundness and completeness of oracles. Because oracles (and their approximations) are typically computationally expensive, an approach to the provision of oracle information may use a probabilistic approach even where an complete and precise answer is possible. For a definite oracle, it responds with either a 1 or a 0 showing that the test activity sequence is acceptable or unacceptable respectively, for each sequence of test activities. It may also be useful to generalise this definition to allow for probabilistic oracles:

A Probabilistic Oracle $D\tilde{}$ is a function from a test activity sequence to [0, 1], i.e., $D\tilde{}: I \rightarrow [0, 1]$. A probabilistic oracle results a real number in the closed interval [0, 1], showing a less precise response than a definite oracle. As with definite oracles, we do not need a probabilistic oracle to be a total function. A probabilistic oracle can be used to represent the case where the oracle is only able to offer a probability that the test case is acceptable, or for other situations where some degree of imprecision is to be tolerated in the oracle's response. Since a definite oracle is only a special case of a probabilistic oracle, we use the term oracle, to refer to either definite or probabilistic oracles where the terminology applies to both.

We can relax our definition of soundness to cater for probabilistic oracles:

A Probabilistic Oracle, $D\tilde{}$ is sound iff $D_\sigma \in [0, 0.5)$ when $G_\sigma = 0$ and $D_\sigma \in [0.5, 1]$ when $G_\sigma = 1$.

IV. Conclusion

We rest our presentation in this paper with the fact that without documenting the mapping between functional requirements with the test cases it is not possible to implement a robust test oracle. Our paper also discussed the future direction in this area. Researchers in this domain are supposed to work with hybrid mechanisms so as to reuse the test oracles and also take a head on challenge to automate the test oracles which will open up a new world in software testing. In a technological world in which software test oracles become more prevalent, it will be crucial for the software testers to be able to evaluate the properties offered by alternative test oracles. This paper meets such requirement upto an appreciable extent.

References

[1] Sheeva Afshan, Phil McMinn, and Mark Stevenson. Evolving readable string test inputs using a natural language model to reduce human oracle cost. In International Conference on Software Testing, Verification and Validation (ICST 2013). IEEE, March 2013.

[2] Wasif Afzal, Richard Torkar, and Robert Feldt. A systematic review of search-based testing for non-functional system properties. Information and Software Technology, 51(6):957–976, 2009.

[3] Bernhard K. Aichernig. Automated black-box testing with abstract VDM oracles. In SAFECOMP, pages 250 259. Springer-Verlag, 1999.

[4] Shay Artzi, Michael D. Ernst, Adam Kiezun, Carlos Pacheco, and Jeff H. Perkins. Finding the needles in the haystack: Generating legal test inputs for object-oriented programs. In 1st Workshop on Model-Based Testing and Object-Oriented Systems (M-TOOS), Portland, OR, October 23, 2006.

[5] F. T. Chan, T. Y. Chen, S. C. Cheung, M. F. Lau, and S. M. Yiu. Application of metamorphic testing in numerical analysis. In Proceedings of the IASTED International Conference on Software Engineering, pages 191–197, 1998.

All rights reserved by www.ijsrd.com