Better Testing Through Oracle Selection (NIER Track)*

Matt Staats, Michael W. Whalen, and Mats P.E. Heimdahl
Dept. of Comp. Sci. and Eng. U of Minnesota
[staats,whalen,heimdahl]@cs.umn.edu

ABSTRACT

In software testing, the test oracle determines if the application under test has performed an execution correctly. In current testing practice and research, significant effort and thought is placed on selecting test inputs, with the selection of test oracles largely neglected. Here, we argue that improvements to the testing process can be made by considering the problem of *oracle selection*. In particular, we argue that selecting the test oracle and test inputs *together* to complement one another may yield improvements testing effectiveness. We illustrate this using an example and present selected results from an ongoing study demonstrating the relationship between test suite selection, oracle selection, and fault finding.

Categories and Subject Descriptors

D.2.5 [**Software Engineering**]: Testing and Debugging

General Terms

Testing

Keywords

test oracles, empirical studies

1. INTRODUCTION

Testing is a critical component of the validation and verification (V&V) of software systems. Testing requires two key components: the *test data* and the *test oracle*. The test data are the inputs given to the application under test (AUT) and the test oracle is the artifact used to determine if the AUT executes correctly [11]. Clearly, both the test data and test oracle contribute to the effectiveness of the testing process; we cannot detect a fault if the test data does not drive execution to a state where the fault manifests itself, nor can we detect a fault if the test execution encounters the fault but the test oracle does not monitor any variables where the fault is revealed as a failure. While there is a substantial body of work related to the effectiveness of test data selection criteria and methods of generating test data [3, 7, 9], considerably less work has been done on the effectiveness of test oracles and their impact on the testing process [4, 16, 11, 12]. Consequently, while we know that the choice of test oracle influences the testing process, we have few, if any techniques for selecting the test oracle, and little understanding of how our selection impacts the testing process.

We believe that this lack of oracle selection techniques represents an opportunity to improve testing. This belief stems from earlier work of the effectiveness of test coverage metrics in which we noted that stronger test oracles sometimes led to dramatic gains in testing effectiveness [10], and more recent work demonstrating theoretically that the test oracle impacts the effectiveness of the testing process [12]. Accordingly, we have begun investigating how the choice of test oracle influences the effectiveness of the testing process, uncovering a number of observations. Of particular note is that it may be possible to improve the effectiveness of testing by selecting both test suites and test oracles *together*—in other words, by effectively pairing test inputs with test oracles.

We present a subset of the results from a recently conducted empirical study within the domain of avionics. As a first step towards a method for effective oracle selection, this study is designed to improve our understanding of the problem. In particular, we studied (1) the influence of test oracles on fixed test suites and (2) the joint influence of test suite size and oracle selection. Our study yields two observations key to this report.

First, an oracle using output variables in conjunction with internal state variables often performs considerably better than an oracle using only the output variables (up to 27% improvement), with diminishing potential gains being observed as an increasing number of internal state variables are included.

Second, the relationship between test suite size, the number of variables observed by the oracle, and fault finding varies considerably between case examples. In particular, for some case examples it appears that both increasing test suite size and considering internal state information are reasonable means of improving testing effectiveness, while for other case examples considering internal state information does not appear as useful.

These observations indicate that developing effective methods of pairing test inputs and test oracles is an achievable goal. In the remainder of the paper, we will argue why selecting test inputs and oracles together makes conceptual sense, and present the empirical data supporting this argument.

2. RELATED WORK

Existing research on test oracles relates primarily to addressing the "oracle problem" or "oracle assumption"—the assumption that developers can determine the correctness of produced output de-

*This work has been partially supported by NASA Ames Research Center Cooperative Agreement NNA06CB21A and the National Science Foundation under Grant No. CNS-0931931, CCF-0916583, and CNS-0821474. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the sponsoring organizations.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
ICSE'11, May 21–28, 2011, Waikiki, Honolulu, HI, USA
Copyright 2011 ACM 978-1-4503-0445-0/11/05 ...$10.00

spite the apparent difficulty of doing so—by developing test oracles based on methods other than concretely specified oracles [14]. Baresi and Young evaluate several examples of these (and other) approaches to addressing the oracle problem [2].

In our domain of interest, however, concrete oracles with expected values manually specified by users are generally used. We are therefore interested in determining which variables should be considered by such an oracle. Our study explores how the selection of these variables influences the effectiveness of the testing process. To the best of our knowledge, only two other works (by Memon et al. and Briand et al. [16, 4]) empirically study oracle selection.

Memon et al. define oracles for GUI systems using combinations of oracle information and procedures, and examine their fault finding effectiveness using mutation testing [16]. Our work differs primarily in the method used to vary the oracle (we do not vary the oracle procedure), the analyses performed, and the domain in which the studies were conducted. In particular, while test cases of varying length are explored by the authors, the joint influence of test suite and oracle is not explored in the same manner or detail.

Briand et al. demonstrate that in the domain of object-oriented systems: (1) a concrete, precise oracle outperforms a state-based oracle defined as state invariants for the case examples used, (2) there exist faults only detectable when using both precise oracles and rigorous coverage metrics, and (3) the cost effectiveness of precise oracles varies from system to system [4]. Our work differs primarily in the method used to vary the oracle, the analysis performed, and the domain in which the studies were conducted.

Voas has developed the propagation, infection, and execution technique (PIE) for computing testability metrics for locations in source code. Of relevance here is Voas and Miller's proposal to determine where faults may be masked using PIE and monitor internal state accordingly (using assertions) [13]. As PIE essentially performs an empirical study to determine masking, it is similar to running our empirical study to find internal variables to monitor. However, no large scale empirical study has been conducted to determine the efficacy of using the PIE technique for oracle selection.

3. TEST ORACLES

We use the definition presented by Richardson et al. that divides test oracles into two parts: the *oracle information* and the *oracle procedure* [11] . The oracle information specifies what is considered correct behavior, and the oracle procedure verifies that test executions correspond to the oracle information.

Our case examples are all avionics systems developed in Simulink [8] and translated to Lustre [6]. Simulink and Lustre are examples of *synchronous dataflow languages* [6]. A test input for a synchronous reactive system is defined as a sequence of steps, with each step specifying a value for each input variable. This produces a corresponding sequence of values of the system's internal state variables and outputs. In our study, the oracle information is specified as sequences of test inputs with corresponding expected changes to the AUT output variables and internal state variables, i.e., input/expected value(s) pairs. The oracle procedure simply matches the actual results from the AUT with the oracle information. We term oracles constructed in this fashion as *expected value oracles*. In our experience with industrial partners, expected value oracles are commonly used in testing synchronous reactive systems in the avionics domain.

A key property of expected value oracles is the subset of outputs and internal state variables considered by the oracle information; variables that we term the *oracle data*. Currently, from our observations, the prevalent oracle data used in testing critical systems consists of all of the outputs. We term such an oracle an *output-only* oracle. We are interested in how the addition of internal state information to the oracle data influences the effectiveness of the testing process. We term oracles consisting of all the outputs and one or more internal state variables *output-base* oracles, and term an oracle considering all outputs and internal state variables the *maximum* oracle. We explore how the addition of internal state variables influences the effectiveness of the testing process by varying the oracle data, ranging from the output-only oracle to the *maximum* oracle.

4. PAIRING TEST DATA WITH ORACLES

Consider the process of testing a system. Suppose we plan to use two test suites: (1) a test suite generated to meet the Modified Condition/Decision Coverage (MCDC) criterion [5], and (2) a test suite generated to meet the Unique First Cause (UFC) coverage criterion [15], a black box test coverage criterion based on exercising Linear Temporal Logic (LTL) requirements. Now consider developing a test oracle for these test suites, and ask: *what should the oracle data be?* One common answer in the domain of avionics, and perhaps in software engineering research in general, is to simply defined expected values for all of the outputs, irrespective of the test suite used.

For $MCDC$, the tests are designed to exercise internal code structures. Accordingly, it is possible for each test to be very short, with its impact limited to the Boolean decision it is designed to exercise. In contrast, for UFC, the tests are designed to exercise functional requirements. These requirements are defined in terms of inputs and outputs, and tests, thus, tend to be longer, as they must typically cause the system to exhibit a sequence of behaviors.

Given the differences between $MCDC$ and UFC, can we not tailor a test oracle to their varying characteristics? It seems reasonable to use a test oracle considering internal state information when using $MCDC$, as the tests are designed specifically to manipulate internal behavior, and might not cause information to propagate to outputs. When using UFC, however, the tests are designed to test black-box behavior, and, thus, seem more likely to propagate information to the outputs—considering internal state information seems less less valuable when using UFC.

In this example, we have paired a single test oracle with a test suite. However, we can take this idea to its logical extreme by considering pairing a separate test oracle with each test input. For example, if a test input T is designed to exercise a branch deep in the source code, we could consider only variables related to this branch when running test T. When running a test T' exploring a different branch deep in another portion of the source code, we might consider a completely different separate set of variables.

5. STUDY AND RESULTS

In our study, we were interested in how the use of internal state information in oracle data influences the effectiveness of the testing process, as measured by the number of faults detected (as defined by [1]).

5.1 Experimental Setup Overview

We used four industrial synchronous reactive systems developed by Rockwell Collins Inc. in our experiment. Two systems, *DWM_1* and *DWM_2*, represent portions of a Display Window Manager (DWM) for a commercial cockpit display system. Two systems represent various aspects of a Flight Guidance System (FGS), which is a component of the overall Flight Control System (FCS) in a commercial aircraft. The two FGS systems in this paper focus on the mode logic of the FGS. The *Vertmax_Batch* and *Latctl_Batch* systems describe the vertical and lateral mode logic for a flight

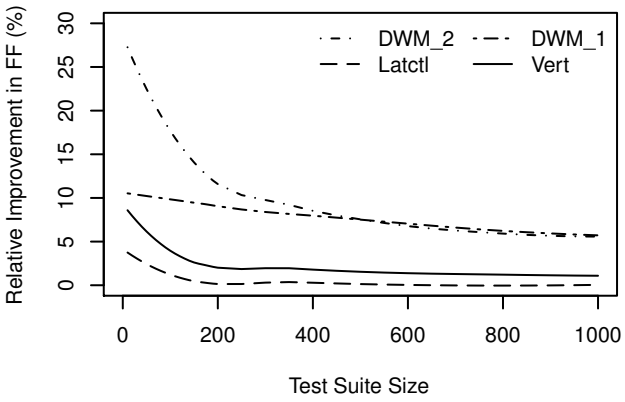


Figure 1: Relative FF Improvement Across Test Sizes

guidance system. All represent sizable, operational systems.

For each case example, we performed the following steps:

1. Generated random test data: We generated random inputs for 1,000 tests of test lengths between 2-10 steps.

2. Generated mutants: We generated 200 mutants, each containing a single fault, and removed functionally equivalent mutants.

3. Ran test suite on mutants: We ran each mutant and the original case example using the entire randomized test suite and collected the internal state and output variable values produced at every step. This yields raw data used for the remaining steps in our study.

4. Generated oracles: We generated a sets of oracles ranging, in size from the output-only oracle to the maximum oracle. We randomly selected which internal variables were to be included in an oracle. We also generated all oracles of size 1 (i.e., observing 1 variable).

5. Generated test suites: We randomly generated 36 subsets of the randomized test suite, with subsets containing 10 to 1,000 tests from the original test suite.

6. Assessed fault finding ability of each oracle and test suite combination: We determined how many mutants were detected by every oracle and a reduced test suite combination. Note that the full results of our analysis are available online.

These steps are routine in experiments based on mutation testing; in this case, the details are similar to the steps used in [10].

One relevant risk of using mutation testing in examining these questions is *functionally equivalent* mutants, in which faults exist but these faults cannot cause a *failure*, which is an externally visible deviation from correct behavior. For our study, we used model checking to detect and remove functionally equivalent mutants. This is made possible due to our use of synchronous reactive systems as case examples—each system is finite, and, thus, determining equivalence is decidable.

After conducting our study, we performed a number of analyses. Here we examine those analyses that we believe support the idea of pairing test inputs and test oracles.

5.2 Improvement Using Maximum Oracle

In Figure 1 we plot the relative improvement in fault finding when using the maximum oracle over the output-only oracle for each test suite (smoothed with LOESS using a smoothing factor of 0.70). In Table 1, we list the oracle sizes for each oracle and the relative increase in oracle size when using the maximum oracle.

For each case example, we can see that the relative improvement when using a maximum oracle starts fairly high, between 4% and 27%, but drops off quickly as we near 200 tests, dropping below 10% for all case examples at 1,000 tests. This indicates that for a small number of tests, the inclusion of internal state information into the test oracle is potentially highly influential on effectiveness, but less so for larger number of tests.

	OO Size	MX Size	Rel. Size Inc.
DWM_1	9	53	489%
DWM_2	7	130	1757%
Latctl_Batch	1	23	2200%
Vertmax_Batch	2	79	3850%

Table 1: Oracle Sizes
(OO = Output-Only Oracle, MX = Maximum Oracle)

We can also see the maximum oracle is between 4.8 and 133 times larger than the output-only oracle—a substantial increase in oracle size for potentially modest gains in fault finding. This highlights the need for effective methods of oracle selection: the fault finding improvements are desirable, particularly in the domain of critical systems, but using the maximum oracle is likely more expensive than using the maximum output-only oracle. To gain these improvements without incurring potentially unacceptable increases in testing cost, we must develop methods of determine which combination of variables is most effectively used in an oracle.

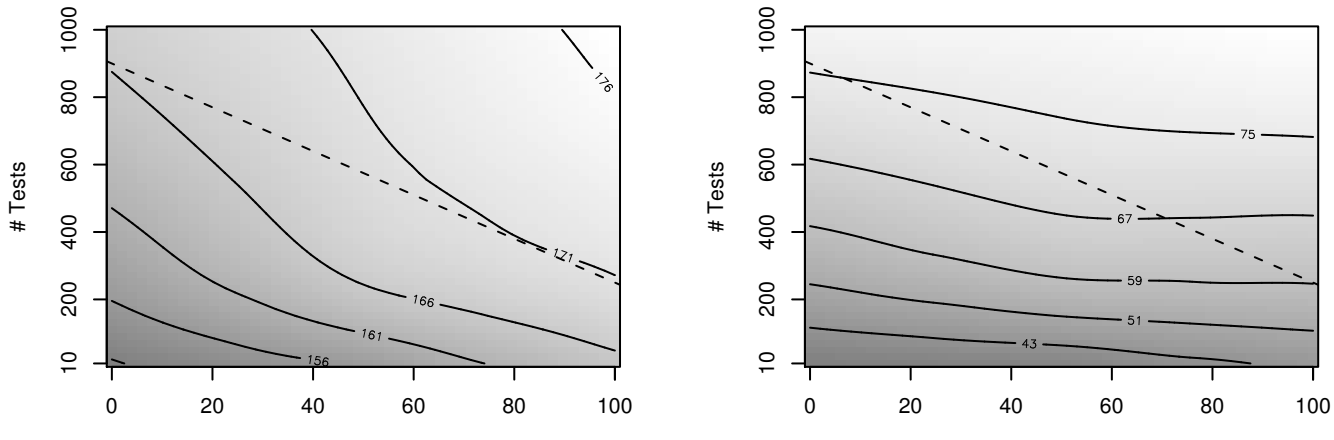
Note that for several of the test-oracles/test-suite combinations, we manually examined how internal state improved the effectiveness of testing, and found that each internal variable reveals a small number of additional faults not caught by the output-only oracle, with many internal variables revealing no additional faults. Furthermore, the set of additional faults caught is nearly always disjoint between variables. Thus, while the maximum oracle is considerably larger than the output-only oracle, only a small portion of the added internal state variables contribute to improving testing effectiveness.

5.3 Test Suites, Oracles, and Fault Finding

We present the relationship between test suite size, oracle size and fault finding as a contour map in Figure 2 for the *DWM_1* and *DWM_2* case examples. As with Figure 1, these figures have been smoothed using LOESS. In these figures, the light areas represent combinations of test suites and oracles that reveal a relatively high number of faults, and dark areas represent combinations of test suites and oracles that reveal a relatively low number of faults. Each contour line represents a constant level of fault finding. (The straight dashed line is explained below.) For example, we can see that for the *DWM_1* system, 800 tests and the largest output-only oracle reveals roughly the same number of faults as 200 tests and an output-base oracle containing 60% of the internal variables (166 faults), whereas for the *DWM_2* system, 800 tests and and the largest output-only oracle reveals more faults than even 600 tests with the maximum oracle.

Given this information and a suitable cost function, we can determine what pairing of test suite and oracle maximizes the number of faults revealed. For example, assume that the testing costs are such that using a test suite of size x with the maximum oracle and a test suite of size $3x$ with the output-only oracle cost the same. Furthermore, assume that the testing resources allocated allow us to use 900 tests with the output-only oracle. Plotting this cost function yields the straight dashed lines present in Figure 2. Examining these lines, we can determine that the most effective, affordable test suite and oracle combination for the *DWM_1* system is 410 tests with an oracle containing 80% of internal variables, whereas the most effective, affordable combination for the *DWM_2* system is 900 tests with the output-only oracle.

As shown, given a set cost function and finite resources, the most effective test suite and oracle combination can vary depending on



(a) DWM 1 Contour Map (Output-Base)

(b) DWM 2 Contour Map (Output-Base)

Figure 2: Combined Effect of Oracle Size and Test Suite Size on Fault Finding

how test suite and oracle size jointly influence the number of faults revealed. For example, if we were to reverse our test suite and oracle selections in the previous paragraph—i.e., use 900 tests with the output-only oracle for the *DWM_1* system and 300 tests with the maximum oracle for the *DWM_2* system—we would find approximately 5.6% fewer faults for the *DWM_1* system and 11.1% fewer faults for the *DWM_2* system.

6. CONCLUSION

We believe this study demonstrates an opportunity to improve testing effectiveness through better selection of oracle data. However, given the large variability between case examples, determining how to improve the test oracle is non-obvious and precludes general guidelines. This brings us to the goal of this paper: motivating the need for further study on test oracles. Our initial results indicate that careful selection of the test oracle, perhaps considering the test suite used, may improve testing. Nevertheless, our results also indicate this is a challenge, requiring future study. We therefore propose the following challenges for future research:

- First, for a given method of selecting oracles and test suites, how can we efficiently determine the most effective combination of test suite and oracle for a given system?
- Second, how does the use of test suites generated to meet coverage criteria (both structural and requirements coverage criteria) influence the effect of oracle selection on the number of faults revealed?
- Finally, is it possible to prioritize tests more effectively given additional information about a program's internal state?

7. REFERENCES

[1] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr. Basic Concepts and Taxonomy of Dependable and Secure Computing. *IEEE Trans. Dependable and Secure Computing*, 1(1):11–33, 2004.

[2] L. Baresi and M. Young. Test oracles. In *Technical Report CIS-TR-01-02, Dept. of Computer and Information Science, Univ. of Oregon*.

[3] A. Bertolino. Software testing research: Achievements, challenges, dreams. In L. Briand and A. Wolf, editors, *Future of Software Engineering 2007*. IEEE-CS Press, 2007.

[4] L. Briand, M. DiPenta, and Y. Labiche. Assessing and improving state-based class testing: A series of experiments. *IEEE Trans. on Software Engineering*, 30 (11), 2004.

[5] J. J. Chilenski and S. P. Miller. Applicability of Modified Condition/Decision Coverage to Software Testing. *Software Engineering Journal*, pages 193–200, September 1994.

[6] N. Halbwachs. *Synchronous Programming of Reactive Systems*. Klower Academic Press, 1993.

[7] M. Harrold. Testing: a roadmap. In *Proc. of the Conf. on the Future of Software Engineering*, pages 61–72. ACM New York, NY, USA, 2000.

[8] Mathworks Inc. Simulink product web site. http://www.mathworks.com/products/simulink.

[9] M. Pezze and M. Young. *Software Test and Analysis: Process, Principles, and Techniques*. John Wiley and Sons, October 2006.

[10] A. Rajan, M. Whalen, and M. Heimdahl. The effect of program and model structure on MC/DC test adequacy coverage. In *Proc. of the 30th Int'l Conference on Software engineering*, pages 161–170. ACM New York, NY, USA, 2008.

[11] D. J. Richardson, S. L. Aha, and T. O'Malley. Specification-based test oracles for reactive systems. In *Proc. of the 14th Int'l Conference on Software Engineering*, pages 105–118. Springer, May 1992.

[12] M. Staats, M. Whalen, and M. Heimdahl. Programs, tests, and oracles: The foundations of testing revisited. In *Proc. of the Int'l Conf. on Software Engineering 2011*, 2011.

[13] J. Voas and K. Miller. Putting assertions in their place. In *Software Reliability Engineering, 1994., 5th Int'l Symposium on*, pages 152–157, 1994.

[14] E. Weyuker. The oracle assumption of program testing. In *13th Int'l Conf on System Sciences*, pages 44–49.

[15] M. Whalen, A. Rajan, and M. Heimdahl. Coverage metrics for requirements-based testing. In *Proceedings of International Symposium on Software Testing and Analysis*, pages 25–36. ACM, July 2006.

[16] Q. Xie and A. Memon. Designing and comparing automated test oracles for gui-based software applications. *ACM Trans. on Software Engineering and Methodology (TOSEM)*, 16(1):4, 2007.