The Influence of Multiple Artifacts on the Effectiveness of Software Testing*

Matt Staats
Dept. of Computer Science and Eng.
University of Minnesota
staats@cs.umn.edu

ABSTRACT

The effectiveness of the software testing process is determined by artifacts used in testing, including the program, the set of tests, and the test oracle. However, in evaluating software testing techniques, including automated software testing techniques, the influence of these testing artifacts is often overlooked. In my upcoming dissertation, we intend to explore the interrelationship between these three testing artifacts, with the goal of establishing a *solid scientific foundation* for understanding how they interact. We plan to provide two contributions towards this goal. First, we propose a theoretical framework for discussing testing based on previous work in the theory of testing. Second, we intend to perform a rigorous empirical study controlling for program structure, test coverage criteria, and oracle selection in the domain of safety critical avionics software.

Categories and Subject Descriptors

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

General Terms

Experimentation,Verification

Keywords

empirical studies, theory of testing

1. INTRODUCTION

Software testing is a key component of the validation and verification (V&V) phase of software development. It is a potentially costly and time consuming process, often requiring significant manual effort on the part of developers and testers. Consequently, determining how to test software is both a crucial and common task. As with any process, we are interested in maximizing the effectiveness of the testing process with respect to a specified goal (e.g., fault detection) and without exceeding our allocation of finite resources. It is this drive to maximize effectiveness and minimize cost that determines, or at least should determine, how software is tested. Evaluating proposed testing techniques with respect to these goals is thus a key task in testing research.

There are several testing artifacts that contribute to the overall effectiveness of the the testing process. Much of testing research (perhaps even most testing research) is focused on determining what *test inputs* (often referred to as simply *tests*) to use, e.g. creation and evaluation of test coverage criteria or automatic test generation tools. However, in previous work, we and other researchers have noticed that several other artifacts influence the effectiveness of the testing process. Specifically, we have noticed that that the structure of the *program* and the *test oracle* selected influence the results. We outline specific related work in Section 2.

This leads to the question: *how do these artifacts interact to influence the effectiveness of the testing process?* In Figure 1, we illustrate how these three artifacts may interact, along with a fourth artifact, the *specification*, which is generally assumed in some form in most testing research. Unfortunately, much of, if not most of testing research does not explicitly consider how these artifacts interact. Depending on the impact of these artifacts, failing to consider them may lead to conclusions that are misleading, or even incorrect. Consequently, we argue that the influence of these artifacts should be considered when evaluating testing techniques.

Our overall *research objective* is to provide an impetus for testing researchers to consider how testing artifacts may jointly influence the effectiveness of the testing process. There are two primary reasons for this interest. First, as with all empirical research, effective empirical testing research requires us to be aware of factors influencing our results. If we do not control for or at least understand the factors underlying our results, our conclusions will be at best limited in their generalizability and at worst incorrect. Second, we may derive new testing techniques or improve existing techniques through a better understanding of the artifacts influencing the effectiveness of testing.

Of course, this is only a broad goal, not a problem that can be "solved" in a definitive fashion. In our proposed research, we intend to work towards a *solid scientific foundation* focused on the interrelationship between four testing artifacts: tests, programs, oracles, and specifications. We intend to provide two primary contributions towards this foundation: (1) a theoretical framework for discussing testing, and (2) a rigorous multifactor empirical study on test effectiveness, performed within the domain of safety critical avionics systems.

The first contribution is intended to satisfy two goals. First, provide a conceptual framework facilitating discussion. In early work on the theory of testing, the definitions of test data adequacy crite-

*This work has been partially supported by NASA Ames Research Center Cooperative Agreement NNA06CB21A, NASA IV&V Facility Contract NNG-05CB16C, and NSF grants CCF-0916583 and CNS-0931931.

ASE'10, September 20–24, 2010, Antwerp, Belgium.

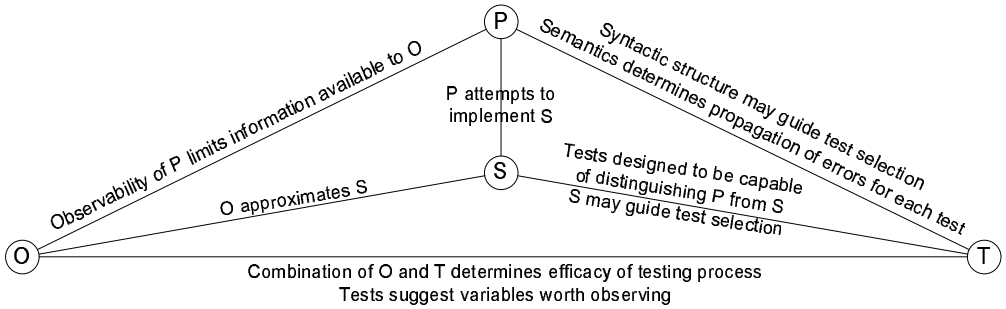

Figure 1: Example Relationships Between Testing Artifacts

ria and correctness were specified. These definitions helped define the problems and ideal solutions of test data selection, and helped to motivate subsequent testing research, both theoretical and empirical. By revisiting this work – extending it to better account for certain overlooked aspects of testing (like test oracles) as needed – we believe we can improve the testing discourse by making our discussions more precise. As an added secondary benefit, further exploring the theory of testing may also provide theoretical insights into the relationship between test selection, oracle selection, and program structure as it pertains to the effectiveness of testing.

The second contribution is intended to provide a rigorous empirical foundation exploring how test data selection, program structure and oracle selection interact and influence the effectiveness of testing. To perform this study, we will use systems from the domain of safety critical avionics systems. We have selected this domain as such systems are amenable to program restructuring, and test oracles for such programs can be defined in a straightforward method. Furthermore, such systems have a high cost of failure and software verification is often very expensive for such systems. Consequently, even minor observed effects are likely to be deemed practically significant. We stress that we do not intend to *propose* any technique; we are instead laying down a rigorous empirical foundation that quantifies the relationship between these testing artifacts. Given such a foundation, later work can then attempt to develop techniques to leverage these interrelationships to improve the quality of automated testing.

2. RELATED WORK

Goodenough and Gerhart's seminal work in [10] outlines a theory of testing based on test data selection. Subsequent work by Weyuker et al. [25] and Gourlay [11] highlight problems in this theory; nevertheless, Goodenough and Gerhart's ideal of a testing criterion capable of finding all faults in a program captures the general goal of test selection, and subsequent theories of program testing generally focus on methods of test selection. In [11], Gourlay presents a mathematical framework for testing, and uses it to reinterpret several previous works.

There exists a wide body of formal and semi-formal work on the specific problem of test selection criteria. Weyuker et al. proposes a set of axioms for test data adequacy criteria for characterizing "good" test data adequacy criteria [23]. This idea is further discussed by Zhu and Hall in [28, 29], by Parrish and Zweben in [16, 17], and by Weyuker again in [24]. Formal analysis of methods of comparing test selection criteria has been performed by many authors, including Weyuker, Weiss and Hamlet in [26], and by Frankl and Weyuker (specifically for partition testing methods) in [8].

Several theories of program testing based on test selection exist. Howden introduces a theory of functional testing in [13], which views programs and the synthesis of other functions. Morell introduces a theory of fault-based testing in [15], in which the goal of testing is to show the absence of a certain set of faults. Hamlet presents an outline of a theory of software dependability in [12], arguing that the foundations of software testing should be statistical in nature. Zhu and He propose a theory of behavior observation for testing concurrent software systems in [30].

Bernot, Gaudel and other authors have developed a theory of testing based on formal specifications [9, 3]. In this work, they define a *testing context* as a triple (H, T, O) where H is a set of testing hypotheses (i.e., assumptions) about the program and specification, T is a set of tests, and O is a test oracle. This body of work is in terms of algebraic sorts and is not applicable to most real-world testing situations; furthermore, the formalization is quite terse.

Rajan et al. explore the effect of program structure on the effectiveness of the MC/DC coverage criterion [18]. Briand et al. demonstrate (among other things) that in the domain of object-oriented systems, concrete, precise oracles are more effective than oracles defined as state-based invariants [6]. Memon and Soffa explores how various oracles influence the effectiveness of GUI testing [14]. Voas et al., through work on the PIE method, explores how program structure can influence the effectiveness of the testing process and influence oracle selection [20, 21].

Several authors have criticized the current state of software engineering research. In [2], Basili notes the need for empirical studies in computer science. Briand notes that testing research tends to fail to consider confounding factors such as program structure [5]. Bertolino notes the need for a "universal test theory" and a strong body of empirical evidence in testing research [4].

3. PROPOSED INVESTIGATION

We intend to (1) establish a framework describing testing in which to outline and discuss questions of interest, and (2) perform a rigorous empirical study exploring these questions. We are not interested in proposing new testing techniques; instead, we wish to lay the foundation needed to better understand the interaction between multiple testing artifacts. This study will be conducted in the domain of synchronous reactive systems and will consider a select number of test coverage criteria that are commonly considered in this domain.

3.1 Theoretical Framework

Unlike previous work in the theory of testing, we are not primarily concerned with *proving* things. Instead, we are interested in using a theoretical framework for testing to provide a conceptual framework for discussion. Based on our own experiences and the work of others, we are interested in how three testing artifacts interact in testing: test inputs, program structure, and oracle selection. Consequently, any framework must allow us to discuss these three artifacts. Generally, a *specification*, or some notion of correctness, is usually assumed in testing research and thus should also be con-

sidered. Note this yields four artifacts to consider in our theoretical framework, while only three are to be considered in our empirical study.

Rather than invent a completely new framework, we believe it is best to adopt an existing framework, extending and modifying it as necessary. Beginning with Goodenough and Gerhart's seminal work [10], a significant portion of the research in the theory of testing has used a functional model for testing. As a basis for our own work, we have selected Gourlay's framework [11]. We have selected this framework for several reasons, including: (1) Gourlay's framework forms the basis for a significant quantity of relevant work, and thus easily allows us to reexamine previous work in the theory of testing, and (2) it is easy to understand and mostly matches our intuitive sense of the testing process.

3.1.1 Gourlay's Framework

In Gourlay's approach, a *testing system* is defined as a collection $(P, S, T, corr, ok)$ where:

- S is a set of specifications
- P is a set of programs
- T is a set of test inputs (referred to as tests hereafter)
- $corr \subseteq P \times S$
- $ok \subseteq T \times P \times S$

Each specification $s \in S$ represents an abstract, perfect notion of correctness; it is the only arbiter of correctness and represents the true requirements of the software—not the possibly flawed requirements as captured in a document or formal specification.

The predicate $corr$ is defined such that for $p \in P, s \in S$, $corr(p, s)$ holds when p is correct with respect to s. Of course, the value of $corr(p, s)$ is generally not known; this predicate is thus theoretical and used to explore how testing relates to correctness. The predicate ok is defined such that for $p \in P, s \in S, t \in T$ $ok(t, p, s)$ implies that p is judged as correct with respect to specification s for test t. Furthermore, ok is defined such that $\forall p \in P, \forall s \in S, \forall t \in T$ $corr(p, s) \Rightarrow ok(t, p, s)$, i.e., if p is correct with respect to s then ok is true for all tests. This predicate approximately corresponds to what is now called a *test oracle* or simply *oracle*.

3.1.2 Proposed Extension

While intuitively appealing, we have identified two issues with this framework. First, each testing system has only one possible oracle (ok). However, just as there exist many possible tests and programs, there exist many possible oracles for determining if test executions are successful [19]. Selecting an oracle is the problem of *oracle selection*, and we cannot easily discuss or even formulate this problem using Gourlay's framework. Second, the notion of correctness is very coarse. If for $p \in P$ and $s \in S$, $corr(p, s)$ holds then we know that $\forall t \in T$, $ok(t, p, s)$. However, we may wish discuss correctness on a test-by-test basis, i.e., for test t does $corr(p, s)$ hold. Furthermore, it is generally the case that $corr(p, s)$ does not hold, in which case we know nothing about ok's properties.

Both the inability to discuss oracle selection, and the coarse relationship between program correctness and oracle behavior create ambiguities. We therefore make two major changes to Gourlay's definition of a testing system. First, we remove the predicate ok, replacing it with the set O of oracles. We state that an oracle $o \in O$ is a predicate:

$$o : T \times P \times S \rightarrow \{True, False\}$$

An oracle determines, for a given program, specification, and test if the test passes. (This can be equivalently defined as $o \subseteq TxPxS$; our definition is chosen simply for clarity.) Note that oracles need not relate to correctness, though such properties can be defined by relating the oracle to correctness.

Second, we add a predicate defining correctness with respect to a test $t \in T$. This predicate is:

$$corr_T \subseteq T \times P \times S$$

The predicate $corr_T(t, p, s)$ holds if and only if the specification s holds for program p when running test t. Obviously,

$$\forall p \in P, \forall s \in S, \;\; corr(p, s) \Rightarrow \forall t \in T \; corr_T(t, p, s)$$

3.2 Questions of Interest

We now define several questions of interest in this framework. These questions do not necessarily constitute the final set of questions we will address, but are examples of the type of interactions we wish to better understand. We begin by stating the definition of a test coverage criteria in the framework, borrowing Weiss's definition [22]:

$$T_C : P \times S \times 2^T.$$

The development and evaluation of test coverage criteria has been a significant source of interest in testing research. In our work, we will use several test coverage criteria as a method of test selection.

First, let us explore questions related to program structure and test selection. Previous work indicates the effectiveness of test coverage criteria can depend on the program structure [18]. This phenomenon is poorly understood, however – we know more complex conditions are likely to lead to better tests when using coverage criteria defined in terms of conditions, but we have no ability to predict, given a program, how effective test satisfying a test coverage criteria may be.

Assume we have a set of programs $P = \{p_1, p_2 \ldots p_n\}$ and a specification s. Assume that T is the set of all possible tests for the programs in P. Assume that $\forall p_i, p_j \in P, \forall t \in T, corr_T(t, p_i, s) = corr_T(t, p_j, s)$, i.e., all the programs in P are semantically equivalent with respect to s, but may differ in structure. Finally, assume that we have several test coverage criteria $TC_1, TC_2 \ldots TC_N$. Note that for structural coverage criteria, the set of tests satisfying the criteria can vary depending on $p_i \in P$ [18]. We state the *effectiveness* of testing to be the ability of testing to detect faults with a given set of tests and oracle for one of more programs (we can formally define this, but it is cumbersome).

We ask the following questions:

Predicative Power of Program Structure for TC: For a given TC, to what extent does the structure of each $p_i \in P$ influence the fault finding ability of tests satisfying TC? Can we produce an ordering of the programs based on structure, such that we can predict the effectiveness of at test set t satisfying $TC(p_i, s, t)$ based on p_i's position in the order?

Influence of Program Structure between TC: To what extent does the structure of each $p_i \in P$ influence the fault finding ability of tests satisfying different TC? Are some coverage criteria more strongly influenced by program structure than others? Do there exist TC_1 and TC_2 such that for p_i and p_j, test sets satisfying TC_1 find on average more faults than tests sets satisfying TC_2 for p_i, but not for p_j?

Program Structure Ambivalent TCs: Does there exist a coverage criterion TC whose effectiveness remains roughly constant for all elements of P? In particular, are there structural coverage criteria that are not significantly affected by program structure?

Second, let us explore questions related to test oracles and test selection. In our work, we have noted that the effectiveness of oracles can vary depending on the percentage of system state considered. While this matches our intuition – the more we monitor during testing, the more faults we can find – this is unquantified. We therefore wish to empirically determine to what degree this improved fault finding occurs, and if it is feasible and/or cost effective to improve testing effectiveness through improved oracles.

Assume we have a single program p, a specification s, and a set of possible oracles $O = \{o_1, o_2 \dots o_n\}$ that differ only in the set of variables monitored (i.e., oracles differ in amount of system state considered). Assume that T is the set of all possible tests for p. Assume that we have several test coverage criteria $TC_1, TC_2 \dots TC_N$. Finally, assume that $\forall o \in O, corr_T(t, p, s) \implies o(t, p, s)$. In other words, if the program is correct, the test passes with respect to the oracle, but the oracle need not catch every fault. Thus the oracles in O have varying degrees of effectiveness; this motivates the need to select an oracle.

We ask the following questions:

Variance of Effectiveness Between Oracles: To what extent do the oracles in O vary in fault finding effectiveness with respect to T? To what extent are more effective oracles determined by the percentage of system state considered?

Relationship Between the Test Selection and Oracles: How does the combination of test coverage criterion TC and oracle $o \in O$ influence fault finding? Are some coverage criteria more influenced by the choice of oracles than others?

Existence of Cost-Effective Pairs of Test Sets and Oracles: Clearly the most effective pair of test set and oracle is to use all the tests and an oracle considering all the internal state space, but this is also the most expensive testing process. Do there exist pairs of TC and oracles $o \in O$ achieving similar levels of fault finding while requiring significantly less resources (number of tests and percentage of state space explored)?

Finally, let us explore questions concerning combinations of test oracles and program structures. Intuitively, a relationship exists between test oracles and program structures – the program structure determines the system state that can be monitored. However, we are not aware of any studies exploring the influence of this particular combination of testing artifacts, and we have no strong intuition as to how this interaction will influence testing effectiveness. This leads to the following question:

Influence of Program Structure on Oracle Effectiveness: As the program structure changes, how do the answers to the questions asked about oracles above change?

3.3 Planned Empirical Study

To empirically study the questions above, we must vary three artifacts – the program's structure, the test oracles, and the set of tests used – while measuring the fault finding effectiveness of each combination. This will provide the data to address each of the questions stated above (and many more besides). We will use 4 synchronous reactive systems developed at Rockwell Collins to conduct the study. These provide us data from real avionics systems judged to be representative of the field. Currently, we are planning to use 5 test coverage metrics: two structural coverage metrics, branch coverage and Modified Condition/Decision Coverage (MC/DC) coverage [7]; three requirements coverage metrics, UFC coverage, antecedent coverage, and requirements coverage [27]. Colleagues are currently investigating other applicable coverage metrics; if time and resources permit, we will also use these coverage metrics.

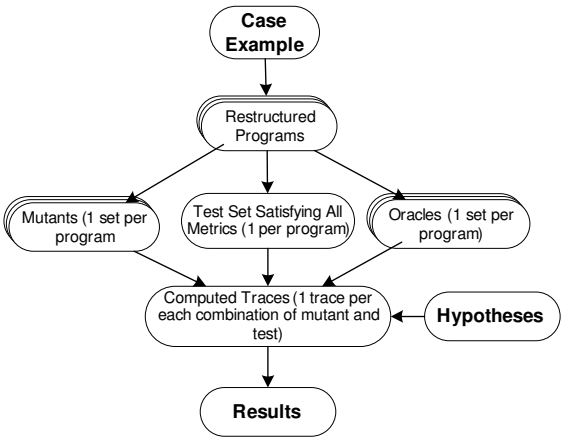

Figure 2: Empirical Study Outline for One Case Example

We illustrate how this study is currently planned in Figure 2. For each system, we start by producing a set of semantically equivalent, but structurally different programs. Methods for producing such programs from Lustre systems were developed in [18]; these primarily operate by modifying the conditions, either increasing the complexity (through inlining) or decreasing the complexity (through outlining). Then, for each program in the set, we produce (1) a set of mutants generated using standard mutation generation techniques [1], (2) sets of tests satisfying various test coverage criteria, and (3) oracles monitoring randomly selected portions of the internal state. Using these artifacts, we then run each test against each mutant, collecting the internal and output state information for every step of every test (i.e., a complete trace of the state information). Using this information, we can compute fault finding ability of many combinations of test sets, oracles, and programs.

Analysis then can proceed using standard statistical tools, aggregating the fault finding measurements generated according to program structures, test coverage criteria, and oracles, or any combination therein, as needed. For example, several of our questions effectively ask if the fault finding measurement of two sets differ, e.g., using coverage criterion C and program structures P_1 and P_2, does the fault finding for P_1 differ from P_2? These type of questions can be easily answered by grouping all the relevant fault finding measurements (in this case, the fault finding measurements collected from using criterion C with program P_1 and P_2), and then testing for a difference in the fault finding measurements of each set using a nonparametric test of significance, such as bootstrapping.

Note that by collecting complete traces of every test against every mutant/system, the data collected is very general and can be quickly reprocessed as questions and hypotheses arise. This is by design – our initial empirical study is intended to provide a foundation for understanding. As we examine the data, we expect new questions will arise. By collecting as much data as possible, we will be able to form and test hypotheses without the need to rerun empirical studies. The downside to this approach is that data collection is very expensive, potentially requiring several days for each system.

4. EXISTING / FUTURE CONTRIBUTIONS

Currently, our existing contributions include:

- Extension of Gourlay's framework for testing to better account for the role of test oracles in the testing process.

- Use of this extended framework to explore desirable properties of oracles, analogous to previous work exploring desirable properties for test coverage metrics.
- Rigorous empirical study conducted over Lustre programs exploring the how the number of variables monitored during random testing influences the fault finding effectiveness. Notable results include the possible existence of cost-effective oracles and the possibility of improving fault finding by selecting better oracles, rather than more tests. These results motivate many of the oracle questions from the previous section.

Future work planned includes:

- Continue to refine framework as needed. Currently, the functional framework outlined suffices to describe the problems of interest. We have considered describing testing using a computational model of testing. Depending on the results of our empirical studies, we may adopt such a model if needed.
- Continue to refine questions of interest. The empirical study is designed such that the data collected will be amenable to exploring many hypotheses, some which will likely be formed only after initial analysis of the data; nevertheless, the initial questions asked will direct our study's setup.
- Finalize empirical study and collect data. The study currently planned is very time and labor intensive, and will generate a *very* large amount of data (estimated at larger than 100 GB). It is imperative that it is well planned.
- Analyze data based on initial questions; form and test hypotheses. Depending on the results of these hypotheses, form and test new hypotheses.

5. REFERENCES

[1] J. Andrews, L. Briand, and Y. Labiche. Is Mutation an Appropriate Tool for Testing Experiments? *Proceedings of the 27th International Conference on Software Engineering (ICSE)*, pages 402–411, 2005.

[2] V. R. Basili and M. V. Zelkowitz. Empirical studies to build a science of computer science. *Commun. ACM*, 50(11):33–37, 2007.

[3] G. Bernot. Testing against formal specifications: A theoretical view. In *TAPSOFT'91: Colloquium on Trees in Algebra and Programming (CAAP'91)*, page 99. Springer, 1991.

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

[5] L. Briand. A critical analysis of empirical research in software testing. In *Empirical Software Engineering and Measurement, 2007. ESEM 2007. First Int'l Symposium on*, pages 1–8, 2007.

[6] 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.

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

[8] P. Frankl and E. Weyuker. A formal analysis of the fault-detecting ability of testing methods. In *IEEE Trans. on Software Engineering*, 1993.

[9] M. Gaudel. Testing can be formal, too. *Lecture Notes in Computer Science*, 915:82–96, 1995.

[10] J. B. Goodenough and S. L. Gerhart. Toward a theory of testing: Data selection criteria. In R. T. Yeh, editor, *Current trends in programming methodology*. Prentice Hall, 1979.

[11] J. Gourlay. A mathematical framework for the investigation of testing. *IEEE Trans. on Software Engineering*, pages 686–709, 1983.

[12] D. Hamlet. Foundations of software testing: dependability theory. *ACM SIGSOFT Software Engineering Notes*, 19(5):128–139, 1994.

[13] W. Howden. Theory and practice of functional testing. *IEEE Software*, 2(5):6–17, 1985.

[14] A. M. Memon and M. L. Soffa. Regression testing of guis. In *In Proceedings of the 9th European Software Engineering Conference*, 2003.

[15] L. Morell. A theory of fault-based testing. *IEEE Transactions on Software Engineering*, 16(8):844–857, 1990.

[16] A. Parrish and S. Zweben. Analysis and refinement of software test data adequacy properties. *IEEE Trans. on Software Engineering*, 17(6):565–581, 1991.

[17] A. Parrish and S. Zweben. Clarifying some fundamental concepts in software testing. *IEEE Trans. on Software Engineering*, 19(7):742–746, 1993.

[18] 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.

[19] 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.

[20] J. Voas and K. Miller. The revealing power of a test case. *Software Testing, Verification and Reliability*, 2(1):25–42, 1992.

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

[22] S. Weiss. Comparing test data adequacy criteria. *ACM SIGSOFT Software Engineering Notes*, 14(6):42–49, 1989.

[23] E. Weyuker. Axiomatizing software test data adequacy. *IEEE Trans. on Software Engineering*, 12(12):1128–1138, 1986.

[24] E. Weyuker. The evaluation of program-based software test data adequacy criteria. *Communications of the ACM*, 31(6):668–675, 1988.

[25] E. Weyuker and T. Ostrand. Theories of program testing and the application of revealing subdomains. *IEEE Trans. on Software Engineering*, pages 236–246, 1980.

[26] E. Weyuker, S. Weiss, and D. Hamlet. Comparison of program testing strategies. In *Proc. of the Symposium on Testing, Analysis, and Verification*, page 10. ACM, 1991.

[27] 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.

[28] H. Zhu. Axiomatic assessment of control flow-based software test adequacy criteria. *Software Engineering Journal*, 10(5):194–204, 1995.

[29] H. Zhu and P. Hall. Test data adequacy measurement. *Software Engineering Journal*, 8(1):21–29, 1993.

[30] H. Zhu and X. He. A theory of behaviour observation in software testing. Technical report, 1999.