On the Danger of Coverage Directed Test Case Generation

Matt Staats[1], Gregory Gay[2], Michael Whalen[2], Mats Heimdahl[2]

[1] Korea Advanced Institute of Science & Technology, Daejeon, Republic of Korea
[2] University of Minnesota, Minneapolis MN, USA
staatsm@kaist.ac.kr, greg@greggay.com, {whalen,heimdahl}@cs.umn.edu

Abstract. In the avionics domain, the use of structural coverage criteria is legally required in determining test suite adequacy. With the success of automated test generation tools, it is tempting to use these criteria as the basis for test generation. To more firmly establish the effectiveness of such approaches, we have generated and evaluated test suites to satisfy two coverage criteria using counterexample-based test generation and a random generation approach, contrasted against purely random test suites of equal size.

Our results yield two key conclusions. First, coverage criteria satisfaction alone is a poor indication of test suite effectiveness. Second, the use of structural coverage as a supplement—not a target—for test generation can have a positive impact. These observations points to the dangers inherent in the increase in test automation in critical systems and the need for more research in how coverage criteria, generation approach, and system structure jointly influence test effectiveness.

1 Introduction

In software testing, the need to determine the adequacy of test suites has motivated the development of several test coverage criteria [1]. One such class of criteria are *structural coverage criteria*, which measure test suite adequacy in terms of coverage over the structural elements of the system under test. In the domain of critical systems—particularly in avionics—demonstrating structural coverage is required for certification [2]. In recent years, there has been rapid progress in the creation of tools for automatic directed test generation for structural coverage criteria [3–5]; tools promising to improve coverage and reduce the cost associated with test creation.

In principle, this represents a success for software engineering research: a mandatory—and potentially arduous—engineering task has been automated. However, while there is some evidence that using structural coverage to guide random test generation provides better tests than purely random tests, the effectiveness of test suites automatically generated to satisfy various structural coverage criteria has not been firmly established. In pilot studies, we found that test inputs generated specifically to satisfy three structural coverage criteria via

counterexample-based test generation were *less effective* than random test inputs [6]. Further, we found that reducing larger test suites providing a certain coverage—in our case MC/DC—while maintaining coverage reduced their fault finding significantly, hinting that it is not always wise to build test suites solely to satisfy a coverage criterion [7].

These results are concerning. Given the strong incentives and the ability to automate test generation, it is essential to ask: *"Are test suites generated using automated test generation techniques effective?"* In earlier studies, we used a single system to explore this question. In this paper, we report the results of a study conducted using four production avionics systems from Rockwell Collins Inc. and one example system from NASA. Our study measures the fault finding effectiveness of automatically generated test suites satisfying two structural coverage criteria, branch coverage and Modified Condition Decision Coverage (MC/DC coverage) as compared to randomly generated test suites of the same size. We generate tests using both counterexample-based test generation and a random generation approach. In our study we use mutation analysis [8] to compare the effectiveness of the generated test suites as compared to purely randomly generated test suites of equal size.

Our results show that for both coverage criteria, in our industrial systems, the automatically generated test suites perform *significantly worse* than random test suites of equal size when coupled with an output-only oracle (5.2% to 58.8% fewer faults found). On the other hand, for the NASA example—which was selected specifically because its structure is significantly different from the Rockwell Collins systems—test suites generated to satisfy structural coverage perform dramatically better, finding 16 times as many faults as random test suites of equal size. Furthermore, we found that for most combinations of coverage criteria and case examples, randomly generated test suites reduced while maintaining structural coverage find *more* faults than pure randomly generated test suites of equal size, finding up to 7% more faults.

We draw two key conclusions from these results. First, automatic test generation to satisfy branch or MC/DC coverage does not, for the systems investigated, yield effective tests relative to their size. This in turn indicates that satisfying even a highly rigorous coverage criterion such as MC/DC is a poor indication of test suite effectiveness. Second, the use of branch or MC/DC as a supplement—not a target—for test generation (as Chilensky and Miller recommend in their seminal work on MC/DC [9]) does appear effective.

These results in this paper highlight the need for more research in how the coverage criterion, test generation approach, and the structure of the system under test jointly influence the effectiveness of testing. The increasing availability and use of advanced test-generation tools coupled with our lack of knowledge of their effectiveness is worrisome and careful attention must be paid to their use and acceptance.

2 Related Work

There exist a number of empirical studies comparing structural coverage criteria with random testing, with mixed results. Juristo et al. provide a survey of much of the existing work [10]. With respect to branch coverage, they note that some authors (such as Hutchins et al. [11]) find that it outperforms random testing, while others (such as Frankl and Weiss [12]) discover the opposite. Namin and Andrews have found coverage levels are positively correlated with fault finding effectiveness [13]. Theoretical work comparing the effectiveness of partition testing against random testing yields similarly mixed results. Weyuker and Jeng, and Chen and Yu, indicated that partition testing is not necessarily more effective than random testing [14, 15]. Later theoretical work by Gutjahr [16], however, provides a stronger case for partition testing. Arcuri et al. [17] recently demonstrated that in many scenarios, random testing is more predictable and cost-effective at reaching high levels of structural coverage than previously thought. The authors have also demonstrated that, when cost is taken into account, random testing is often more effective at detecting failures than a popular alternative—adaptive random testing [18].

Most studies concerning automatic test generation for structural coverage criteria are focused on how to generate tests quickly and/or improve coverage [19, 3]. Comparisons of the fault-finding effectiveness of the resulting test suites against other methods of test generation are few. Those that exist apart from our own limited previous work are, to the best of our knowledge, studies in concolic execution [4, 5]. One concolic approach by Majumdar and Sen [20] has even merged random testing with symbolic execution, though their evaluation only focused on two case examples, and did not explore fault finding effectiveness.

Despite the importance of the MC/DC criterion [9, 2], studies of its effectiveness are few. Yu and Lau study several structural coverage criteria, including MC/DC, and find MC/DC is cost effective relative to other criteria [21]. Kandl and Kirner evaluate MC/DC using an example from the automotive domain, and note less than perfect fault finding [22]. Dupuy and Leveson evaluate the MC/DC as a compliment to functional testing, finding that the use of MC/DC improves the quality of tests [23]. None of these studies, however, compare the effectiveness of MC/DC to that of random testing. They therefore do not indicate if test suites satisfying MC/DC are truly effective, or if they are effective merely because MC/DC test suites are generally quite large.

3 Study

Of interest in this paper are two broad classes of approaches: random test generation and directed test generation. In random test generation, tests are randomly generated and then later reduced with respect to the coverage criterion. This approach is useful as a gauge of value of a coverage criterion: if tests randomly generated and reduced with respect to a coverage criterion are more effective than pure randomly generated tests, we can safely conclude the use of the coverage criterion led to the improvement. Unfortunately, evidence demonstrating

this is, at best, mixed for branch coverage [10], and non-existent for MC/DC coverage.

Directed test generation is specifically targeted at satisfying a coverage criterion. Examples include heuristic search methods and approaches based on reachability [19, 3, 4]. Such techniques have advanced to the point where they can be effectively applied to real-world avionics systems. Such approaches are usually slower than random testing, but offer the potential to improve the coverage of the resulting test suites. We aim to determine if using existing directed generation techniques with these criteria results in test suites more effective than randomly generated test suites. Evidence addressing this is sparse and, for branch and MC/DC coverage, absent from the critical systems domain.[3]

We expect that a test suite satisfying the coverage criterion to be, at a minimum, at least as effective as randomly generated test suites of equal size. Given the central—and mandated—role the coverage criteria play within certain domains (e.g., DO-178B for airborne software [2]), and the resources required to satisfy them, this area requires additional study. We thus seek answers to the following research questions:

RQ1: *Are random test suites reduced to satisfy branch and MC/DC coverage more effective than purely randomly generated test suites of equal size?*

RQ2: *Are test suites directly generated to satisfy branch and MC/DC coverage more effective than randomly generated test suites of equal size?*

We explore two structural coverage criteria: branch coverage, and MC/DC coverage [10, 9]. Branch coverage is commonly used in software testing research and improving branch coverage is a common goal in automatic test generation. MC/DC coverage is a more rigorous coverage criterion based on exercising complex Boolean conditions (such as the ones present in many avionics systems), and is required when testing critical avionics systems. Accordingly, we view it as likely to be an effective criterion—particularly for the class of systems studied in this report.

3.1 Experimental Setup Overview

In this study, we have used four industrial systems developed by Rockwell Collins, and a fifth system created as a case example at NASA. The Rockwell Collins systems were modeled using the Simulink notation and the NASA system using Stateflow [25, 26], and were translated to the Lustre synchronous programming language [27] to take advantage of existing automation. Two of these systems, *DWM_1* and *DWM_2*, represent portions of a Display Window

[3] It has been suggested that structural coverage criteria should *only* be used to determine if a test suite has failed to cover functionality in the source code [1, 13]. Nevertheless, test suite adequacy measurement can always be transformed into test suite generation. In mandating that a coverage criterion be used for measurement, it seems inevitable that some testers will opt to perform generation to speed the testing process, and such tools already exist [24].

Manager for a commercial cockpit display system. The other two systems—*Vertmax_Batch* and *Latctl_Batch*—represent the vertical and lateral mode logic for a Flight Guidance System (FGS). The NASA system, *Docking_Approach*, describes the behavior of a space shuttle as it docks with the International Space Station.

Information related to these systems is provided in Table 1. We list the number of Simulink subsystems, which are analogous to functions, and the number of blocks, which are analogous to operators. For the NASA example developed in Stateflow, we list the number of states, transitions, and variables.

	# Simulink Subsystems	# Blocks
DWM_1	3,109	11,439
DWM_2	128	429
Vertmax_Batch	396	1,453
Latctl_Batch	120	718

	# Stateflow States	# Transitions	# Vars
Docking_Approach	64	104	51

Table 1. Case Example Information

For each case example, we performed the following steps: (1) mutant generation (described in Section 3.2), (2) random and structural test generation (Section 3.3 and 3.4), and (3) computation of fault finding (Section 3.5).

3.2 Mutant Generation

We have created 250 *mutants* (faulty implementations) for each case example by introducing a single fault into the correct implementation. Each fault was seeded by either inserting a new operator into the system or by replacing an existing operator or variable with a different operator or variable. The mutation operators used in this study are fairly typical and are discussed at length in [28]. They are similar to the operators used by Andrews et al. where they conclude that mutation testing is an adequate proxy for real faults [29].

One risk of mutation testing is *functionally equivalent* mutants—the scenario in which faults exist, but these faults cannot cause a *failure* (an externally visible deviation from correct behavior). This presents a problem when using oracles that consider internal state: we may detect failures that can never propagate to the output. For our study, we used NuSMV to detect and remove functionally equivalent mutants for the four Rockwell Collins systems[4]. This is made possible thanks to our use of synchronous reactive systems—each system is finite, and thus determining equivalence is decidable[5].

The complexity of determining non-equivalence for the *Docking_Approach* system is, unfortunately, prohibitive, and we only report results using the output-only oracle. Therefore, for every mutant reported as killed in our study, there

[4] The percentage of mutants removed is very small, 2.8% on average

[5] Equivalence checking is fairly routine in the hardware side of the synchronous reactive system community; a good introduction can be found in [30].

exists at least one trace that can lead to a user-visible failure, and all fault finding measurements indeed measure faults detected.

3.3 Test Data Generation

We generated a single set of 1,000 random tests for each case example. The tests in this set are between 2 and 10 steps (evenly distributed in the set). For each test step, we randomly selected a valid value for all inputs. As all inputs are scalar, this is trivial. We refer to this as a *random test suite*.

We have directly generated test suites satisfying the branch and MC/DC [10, 31] criteria. Several variations of MC/DC exist—for this study, we use Masking MC/DC, as it is a common criterion within the avionics community [31].

For our directed test generation approach, we used counterexample-based test generation to generate tests satisfying branch and MC/DC coverage [19, 3]. In this approach each coverage obligation is encoded as a temporal logic formula and the model checker can be used to detect a counterexample (test case) illustrating how the coverage obligation can be covered. This approach guarantees that we achieve the maximum possible coverage of the system under test. This guarantee is why we have elected to use counterexample-based test generation, as other directed approaches (such as concolic/SAT-based approaches) do not offer such a straightforward guarantee. We have used the NuSMV model checker in our experiments [32] because we have found that it is efficient and produces tests that are both simple and short [6].

Note that as all of our case examples are modules of larger systems, the tests generated are effectively *unit tests*.

3.4 Test Suite Reduction

Counterexample-based test generation results in a separate test for each coverage obligation. This leads to a large amount of redundancy in the tests generated, as each test likely covers several obligations. Consequently, the test suite generated for each coverage criterion is generally much larger than is required to provide coverage. Given the correlation between test suite size and fault finding effectiveness [13], this has the potential to yield misleading results—an unnecessarily large test suite may lead us to conclude that a coverage criterion has led us to select effective tests, when in reality it is the size of the test suite that is responsible for its effectiveness. To avoid this, we reduce each naïvely generated test suite while maintaining the coverage achieved. To prevent us from selecting a test suite that happens to be exceptionally good or exceptionally poor relative to the possible reduced test suites, we produce 50 different test suites for each case example using this process.

Per *RQ1*, we also create tests suites satisfying branch and MC/DC coverage by reducing the random test suite with respect to the coverage criteria (that is, the suite is reduced while maintaining the coverage level of the unreduced suite). Again, we produce 50 tests suites satisfying each coverage criterion.

For both counterexample-based test generation and random testing reduced with respect to a criterion, reduction is done using a simple greedy algorithm.

We first determine the coverage obligations satisfied by each test generated, and initialize an empty test set *reduced*. We then randomly select a test input from the full set of tests; if it satisfies obligations not satisfied by any test input in *reduced*, we add it to *reduced*. We continue until all tests have been removed from the full set of tests.

For each of our existing reduced test suites, we also produce a purely random test suite of equal size using the set of random test data. We measure suite size in terms of the number of total test steps, rather than the number of tests, as random tests are on average longer than tests generated using counterexample-based test generation. These random suites are used as a baseline when evaluating the effectiveness of test suites reduced with respect to coverage criteria. We also generate random test suites of sizes varying from 1 to 1,000. These tests are not part of our analysis, but provide context in our illustrations.

When generating tests suites to satisfy a structural coverage criterion, the suite size can vary from the minimum required to satisfy the coverage criterion (generally unknown) to infinity. Previous work has demonstrated that test suite reduction can have a negative impact on test suite effectiveness [7]. Despite this, we believe the test suite size most likely to be used in practice is one designed to be small—reduced with respect to coverage—rather than large (every test generated in the case of counterexample-based generation or, even more arbitrarily, 1,000 random tests)[6].

3.5 Computing Fault Finding

In our study, we use *expected value oracles*, which define concrete expected values for each test input. We explore the use of two oracles: an *output-only oracle* that defines expected values for all outputs, and a *maximum oracle* that defines expected values for all outputs and all internal state variables. The output-only oracle represents the oracle most likely to be used in practice. Both oracles have been used in previous work, and thus we use both to allow for comparison. The fault finding effectiveness of the test suite and oracle pair is computed as the number of mutants detected (or "killed").

4 Results and Analysis

We present the fault finding results in Tables 2 and 3, listing for each case example, coverage criterion, test generation method, and oracle: the average fault finding for test suites reduced to satisfy a coverage criterion, next to the average fault finding for random test suites of equal size; the relative change in average fault finding when using the test suite satisfying the coverage criteria versus the random test suite of equal size; and the p-value for the statistical

[6] One could build a counterexample-based test suite generation tool that, upon generating a test, removes from consideration *all* newly covered obligations, and randomly selects a new uncovered obligation to try to satisfy, repeating until finished. Such a tool would produce test suites equivalent to our reduced test suites, and thus require no reduction; alternatively, we could view such test suites as pre-reduced.

		Counterexample Generation				Random Generation			
Case Example	**Oracle**	**Satisfying Branch**	**Random of Same Size**	**% Change**	**p-val**	**Satisfying Branch**	**Random of Same Size**	**% Change**	**p-val**
Latctl_Batch	MX	217.0	215.8	0.6%	0.24	238.7	234.4	1.8%	
	OO	82.2	140.3	-41.4%		196.4	189.2	3.8%	
Vertmax_Batch	MX	211.2	175.3	20.5%		219.5	209.7	4.6%	< 0.01
	OO	77.1	101.7	-24.2%		153.5	143.4	7.0%	
DWM_1	MX	195.1	227.9	-14.4%	< 0.01	230.2	227.1	1.4%	
	OO	32.1	77.9	-58.8%		79.8	76.9	3.77%	0.04
DWM_2	MX	202.1	215.5	-6.2%		232.0	225.8	2.7%	< 0.01
	OO	131.9	174.7	-24.5%		200.3	192.3	4.2%	
Docking_Approach	OO	38.1	2.0	1805%		2.0	2.0	0.0%	1.0

Table 2. Average number of faults identified, branch coverage criterion. OO = Output-Only, MX = Maximum

		Counterexample Generation				Random Generation			
Case Example	**Oracle**	**Satisfying MCDC**	**Random of Same Size**	**% Change**	**p-val**	**Satisfying MCDC**	**Random of Same Size**	**% Change**	**p-val**
Latctl_Batch	MX	235.0	241.8	-2.8%		241.5	240.3	0.3%	
	OO	194.2	226.7	-14.4%		218.6	214.6	1.9%	< 0.01
Vertmax_Batch	MX	248.0	239.3	3.6%	< 0.01	248.0	237.1	4.6%	
	OO	147.0	195.5	-24.8%		204.2	191.4	6.7%	
DWM_1	MX	210.0	230.4	-12.8%		230.6	229.5	0.4%	0.048
	OO	44.6	86.6	-48.5%		85.4	86.4	-1.2%	0.6
DWM_2	MX	233.7	232.2	0.7%	0.08	241.9	235.5	2.7%	< 0.01
	OO	196.2	207.0	-5.2%	< 0.01	222.6	213.5	4.3%	
Docking_Approach	OO	37.34	2.0	1750%		2.0	2.0	0.0%	1.0

Table 3. Average number of faults identified, MCDC criterion. OO = Output-Only, MX = Maximum

analysis below. Note that negative values for *% Change* indicate the test suites satisfying the coverage criterion are less effective on average than random test suites of equal size.

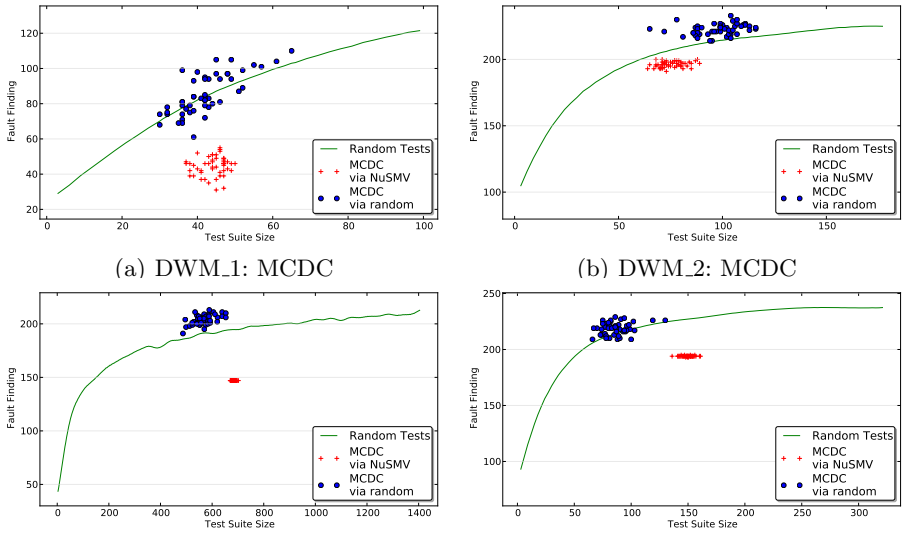
	Branch Coverage	**MCDC Coverage**
DWM_1	100.0%	100.0%
DWM_2	100.0%	97.76%
Vertmax_Batch	100.0%	99.4%
Latctl_Batch	100.0%	100.0%
Docking_Approach	58.1%	37.76%

Table 4. Coverage Achieved (of Maximum Coverage) by Randomly Generated Test Suites Reduced to Satisfy Coverage Criteria

The test suites generated using counterexample-based test generation are guaranteed to achieve the maximum achievable coverage, but the randomly generated test suites reduced to satisfy structural coverage criteria are not. We therefore present the coverage achieved by these test suites (with 100% representing the maximum *achievable* coverage) in Table 4.

In Figure 1, we plot, for MC/DC coverage and four case examples, the test suites size and fault finding effectiveness of every test suite generated when using the output-only oracle.[7] Test suites generated via counterexample-based test generation are shown as pluses, random test suites reduced to satisfy structural coverage criteria are shown as circles, and random test suites of increasing size (including those paired with test suites satisfying coverage criteria) are shown in the line. The line has been smoothed with LOESS smoothing (with a factor

[7] For reasons of space, plots for branch coverage and the maximum oracle are omitted. Figures for the *Docking_Approach* case example are not very illustrative.



(a) DWM_1: MCDC (b) DWM_2: MCDC

(c) Vertmax_Batch: MCDC (d) Latctl_Batch: MCDC

Fig. 1. Faults identified compared to test suite size using NuSMV-generated test suites ('+'), randomly generated test suites reduced to satisfy a coverage criterion ('o'), and pure random test generation (line). Output-only oracles.

of 0.3) to improve the readability of the figure. Note that, while 1,000 random test inputs have been generated, we have only plotted random test suites (i.e., the line) of sizes slightly larger than the test suites satisfying coverage criteria to maintain readability.

4.1 Statistical Analysis

For both *RQ1* and *RQ2*, we are interested in determining if test suites satisfying structural coverage criteria outperform purely random test suites of equal size. We begin by formulating statistical hypotheses H_1 and H_2:

H_1: A test suite generated using random test generation to provide structural coverage will find more faults than a pure random test suite of similar size.
H_2: A test suite generated using counterexample-based test generation to provide structural coverage will find more faults than a random test suite of similar size.
We then formulate the appropriate null hypotheses:
$H0_1$: A test suite generated using random test generation to provide structural coverage will find the same number of faults as a pure random test suite of similar size.
$H0_2$: A test suite generated using counterexample-based test generation to provide structural coverage will find the same number of faults as a random test suite of similar size.

Our observations are drawn from an unknown distribution; therefore, we cannot fit our data to a theoretical probability distribution. To evaluate $H0_1$ and $H0_2$ without any assumptions on the distribution of our data, we use the two-tailed bootstrap paired permutation test (a non-parametric test with no distribution assumptions [33]) with 250,000 samples. We pair each test suite

reduced to satisfy a coverage criteria with a purely random test suite of equal size. We then apply this statistical test for each case example, structural coverage criteria, and test oracle with $\alpha = 0.05$.[8]

4.2 Evaluation of RQ1 and RQ2

Based on the p-values less than 0.05 in Tables 2 and 3, we *reject* $H0_1$ for nearly all case examples and coverage criteria when using either oracle.[9] For cases with differences that are statistically significant, test suites reduced to satisfy coverage criteria are more effective than purely randomly generated test suites of equal size; for these combinations, we accept H_1. Our results confirm that branch and MC/DC coverage can be effective metrics for test suite adequacy within the domain of critical avionics systems: reducing test suites generated via a non-directed approach to satisfy structural coverage criteria is at least not harmful, and in some instances improves test suite effectiveness relative to their size by up to 7.0%. Thus, considering branch and MC/DC coverage when using random test generation generally leads to a positive, albeit slight, improvement in test suite effectiveness.

Based on the p-values less than 0.05 in Tables 2 and 3, we *reject* $H0_2$ for all case examples and coverage criteria when using the output-only oracle. However, for all but one case example, test suites generated via counterexample-based test generation are *less* effective than pure random test suites by 5.2% to 58.8%; we therefore conclude that our initial hypothesis H_2 is false[10]. Nevertheless, the converse of H_2—randomly generated test suites are more effective that equally large test suites generated via counterexample-based test generation—is also false, as the *Docking_Approach* example illustrates. For this case example, random testing is effectively useless, finding a mere 2 faults, while tests generated using counterexample-based test generation find 37-38 faults. We discuss the reasons behind, and implications of, this strong dichotomy in Section 5.

When using the maximum oracle, we see that the test suites generated via counterexample-based test generation fare better. In select instances, counterexample-based test suites outperform random test suites of equal size (notably *Vertmax_Batch*), and otherwise close the gap, being less effective than pure random test suites by at most 14.4%. Nevertheless, we note that for most case examples and coverage criteria, random test suites of equal size are still more effective.

---

[8] Note that we do not generalize across case examples, oracles or coverage criteria, as the needed statistical assumption, random selection from the population of case examples, oracles, or coverage criteria, is not met. The statistical tests are used only demonstrate that observed differences are unlikely to have occurred by chance.

[9] We do not reject $H0_1$ for the *DWM_1* case example when using MC/DC coverage and the output-only oracle, nor do we reject $H0_1$ for the *Docking_Approach* case example.

[10] In our previous work we found the opposite effect [34].

5 Discussion

Our results indicate that for our systems (1) the use of branch and MC/DC coverage as a *supplement* to random testing generally results in more effective tests suites than random testing alone, and (2) the use of branch and MC/DC coverage as a *target* for directed, automatic test case generation (specifically counterexample-based test generation) results in *less* effective test suites than random testing alone, with decreases of up to 58.8%. This indicates that branch and MC/DC coverage are—by themselves—not good indicators of test suite effectiveness. Given the role of structural coverage criteria in software validation in our domain of interest, we find these results quite troublesome.

The lack of effectiveness for test suites generated via counterexample-based test generation is a result of the formulation of these structural coverage criteria, properties of the case examples, and the behavior of NuSMV. We have previously shown that varying the structure of the program can significantly impact the number of tests required to satisfy the MC/DC coverage criterion [35]. These results were linked partly to *masking* present in the systems—some expressions in the systems can easily be prevented from influencing the outputs. This can reduce the effectiveness of a testing process based on structural coverage criteria, as we can satisfy coverage obligations for internal expressions without allowing resulting errors to propagate to the output.

This masking can be a problem; we have found that test inputs generated using counterexample-based generation (including those in this study) tend to be short, and manipulate only a handful of input values, leaving other inputs at default values (`false` or `0`) [6]. Such tests tend to exercise the program just enough to satisfy the coverage obligations for which they were generated and do not consider the propagation of values to the outputs. In contrast, random test inputs can vary arbitrarily in length (up to 10 in this study) and vary all input values; such test inputs may be more likely to overcome any masking present in the system.

As highlighted by the *Docking_Approach* example, however, tests generated to satisfy structural coverage criteria can sometimes dramatically outperform random test generation. This example differs from the Rockwell Collins systems chiefly in its structure: large portions of the system's behavior are activated only when very specific conditions are met. The state space is both deep and contains bottlenecks; exploration of states requires relatively long tests with specific combinations of input values. Thus, random testing is highly unlikely to reach much of the state space. The impact of structure on the effectiveness of random testing can be seen in the coverage of the *Docking_Approach* (only 37.7% of obligations were covered) and is in contrast to the Rockwell Collins systems which—while stateful—have a state space that is shallow and highly interconnected and is, therefore, easier to cover with random testing.

We see two key implications in our results. First, per *RQ1*, using branch and MC/DC coverage as an addition to another non-structure-based testing method—in this case, random testing—can yield improvements (albeit small) in the testing process. These results are similar to those of other authors, for

example, results indicating MC/DC is an effective coverage criterion when used to check the adequacy of manual, requirement-driven test generation [23] and results indicating that reducing randomly generated tests with respect to branch coverage yields improvements over pure random test generation [13]. These results, in conjunction with the results for *RQ2*, reinforce the advice that coverage criteria are best applied after test generation to find areas of the source code that have not been tested. In the case of MC/DC this advice is explicitly stated in regulatory requirements and by experts on the use of the criterion [2, 9].

Second, the dichotomy between the *Docking_Approach* example and the Rockwell Collins systems highlights that while the current methods of determining test suite adequacy in avionics systems are themselves inadequate, some method of determining testing adequacy is needed. While current practice stipulates that coverage criteria should be applied after test generation, in practice, this relies on the honesty of the tester (it is not required in the standard). Therefore, it seems inevitable that at least some practitioners will use automatic test generation to reduce the cost of achieving the required coverage.

Assuming our results generalize, we believe this represents a serious problem. The tools are not at fault: we have asked these tools to produce test inputs satisfying branch and MC/DC coverage, and they have done so admirably; for example, satisfying MC/DC for the *Docking_Approach* example, for which random testing achieves a mere 37.7% of the possible coverage. The problem is that the coverage criteria are simply too weak, which allows for the construction of ineffective tests. We see two possible solutions. First, automatic test generation tools could be improved to avoid pitfalls in using structural coverage criteria. For example, such tools could be encouraged to generate longer test cases increasing the chances that a corrupted internal state would propagate to an observable output (or other monitored variable). Nevertheless, this is a somewhat ad-hoc solution to weak coverage criteria and various tool vendors would provide diverse solutions rendering the coverage criteria themselves useless as certification or quality control tools.

Second, we could improve—or replace—existing structural coverage criteria. Automatic test generation has improved greatly in the last decade, thanks to improvements in search heuristics, SAT solving tools, etc. However, the targets of such tools have not been updated to account for this increase in power. Instead, we continue to use coverage criteria that were originally formulated when manual test generation was the only practical method of ensuring 100% coverage. New and improved coverage metrics are required in order to take full advantage of the improvements in automatic test generation without allowing the generation of inefficient test suites (such as some generated in our study).

6 Threats to Validity

External Validity: Our study has focused on a relatively small number of systems but, nevertheless, we believe the systems are representative of the class of systems in which we are interested, and our results are generalizable to other systems in the avionics domain.

We have used two methods for test generation (random generation and counterexample-based). There are many methods of generating tests and these methods may yield different results. Nevertheless, we have selected methods that we believe are likely to be used in our domain of interest.

For all coverage criteria, we have examined 50 test suites reduced using a simple greedy algorithm. It is possible that larger sample sizes may yield different results. However, in previous studies, smaller numbers of reduced test suites have been seen to produce consistent results [35].

Construct Validity: In our study, we measure fault finding over seeded faults, rather than real faults encountered during development. It is possible real faults would lead to different results. However, Andrews et al. showed that seeded faults leads to conclusions similar to those obtained using real faults [29].

We measure the cost of test suites in terms of the number of steps. Other measurements exist, e.g., the time required to generate and/or execute tests [34]. We chose size to be favorable towards directed test generation. Thus, conclusions concerning the inefficacy of directed test generation are reasonable.

Conclusion Validity: When using statistical analyses, we have attempted to ensure the base assumptions beyond these analyses are met, and have favored non-parametric methods. In cases in which the base assumptions are clearly not met, we have avoided using statistical methods. (Notably, we have avoided statistical inference *across* case examples.)

7 Conclusion and Future Work

The results presented in this paper indicate that coverage directed test generation may not be an effective means of creating tests within the domain of avionics systems, even when using metrics which can improve random test generation. Simple random test generation can yield equivalently sized, but more effective test suites (up to twice as effective in our study). This indicates that adequacy criteria are, for the domain explored, potentially unreliable, and thus, unsuitable, for determining test suite adequacy.

The observations in this paper indicate a need for methods of determining test adequacy that (1) provide a reliable measure of test quality and (2) are better suited as targets for automated techniques. At a minimum, such coverage criteria must, when satisfied, indicate that our test suites are better than simple random test suites of equal size. Such criteria must address the problem *holistically* to account for all factors influencing testing, including the program structure, the nature of the state space of the system under test, the test oracle used, and the method of test generation.

8 Acknowledgements

This work has been partially supported by NASA Ames Research Center Cooperative Agreement NNA06CB21A, NASA IV&V Facility Contract NNG-05CB16C, NSF grants CCF-0916583, CNS-0931931, CNS-1035715, and an NSF graduate

research fellowship. Matt Staats was supported by the WCU (World Class University) program under the National Research Foundation of Korea and funded by the Ministry of Eduation, Science and Technology of Korea (Project No: R31-30007) We would also like to thank our collaborators at Rockwell Collins: Matthew Wilding, Steven Miller, and Darren Cofer. Without their continuing support, this investigation would not have been possible. Thank you!

References

1. H. Zhu and P. Hall, "Test data adequacy measurement," *Software Engineering Journal*, vol. 8, no. 1, pp. 21–29, 1993.
2. RTCA, *DO-178B: Software Considerations In Airborne Systems and Equipment Certification*. RTCA, 1992.
3. S. Rayadurgam and M. P. Heimdahl, "Coverage based test-case generation using model checkers," in *Proc. of the 8th IEEE Int'l. Conf. and Workshop on the Engineering of Computer Based Systems*. IEEE Computer Society, April 2001, pp. 83–91.
4. K. Sen, D. Marinov, and G. Agha, "CUTE: A concolic unit testing engine for C," in *Proc. of the 10th European Software Engineering Conf. / 13th ACM SIGSOFT Int'l. Symp. on Foundations of Software Engineering*. ACM New York, NY, USA, 2005.
5. P. Godefroid, N. Klarlund, and K. Sen, "DART: Directed automated random testing," *PLDI05: Proc. of the 2005 ACM SIGPLAN Conf. on Programming Language Design and Implementation*, 2005.
6. M. P. Heimdahl, G. Devaraj, and R. J. Weber, "Specification test coverage adequacy criteria = specification test generation inadequacy criteria?" in *Proc. of the Eighth IEEE Int'l Symp. on High Assurance Systems Engineering (HASE)*, Tampa, Florida, March 2004.
7. M. P. Heimdahl and G. Devaraj, "Test-suite reduction for model based tests: Effects on test quality and implications for testing," in *Proc. of the 19th IEEE Int'l Conf. on Automated Software Engineering (ASE)*, Linz, Austria, September 2004.
8. Y. Jia and M. Harman, "An analysis and survey of the development of mutation testing," *IEEE Transactions on Software Engineering*, no. 99, p. 1, 2010.
9. J. J. Chilenski and S. P. Miller, "Applicability of Modified Condition/Decision Coverage to Software Testing," *Software Engineering Journal*, pp. 193–200, September 1994.
10. N. Juristo, A. Moreno, and S. Vegas, "Reviewing 25 years of testing technique experiments," *Empirical Software Engineering*, vol. 9, no. 1, pp. 7–44, 2004.
11. M. Hutchins, H. Foster, T. Goradia, and T. Ostrand, "Experiments of the effectiveness of dataflow-and controlflow-based test adequacy criteria," in *Proc. of the 16th Int'l Conf. on Software Engineering*. IEEE Computer Society Press Los Alamitos, CA, USA, 1994.
12. P. Frankl and S. N. Weiss, "An experimental comparison of the effectiveness of the all-uses and all-edges adequacy criteria," in *Proc. of the Symposium on Testing, Analysis, and Verification*, 1991.
13. A. Namin and J. Andrews, "The influence of size and coverage on test suite effectiveness," in *Proc. of the 18th Int'l Symp. on Software Testing and Analysis*. ACM, 2009.
14. E. Weyuker and B. Jeng, "Analyzing partition testing strategies," *IEEE Trans. on Software Engineering*, vol. 17, no. 7, pp. 703–711, 1991.

15. T. Chen and Y. Yu, "On the expected number of failures detected by subdomain testing and random testing," *IEEE Transactions on Software Engineering*, vol. 22, no. 2, 1996.
16. W. Gutjahr, "Partition testing vs. random testing: The influence of uncertainty," *IEEE Transactions on Software Engineering*, vol. 25, no. 5, pp. 661–674, 1999.
17. A. Arcuri, M. Z. Z. Iqbal, and L. C. Briand, "Formal analysis of the effectiveness and predictability of random testing," in *ISSTA*, 2010, pp. 219–230.
18. A. Arcuri and L. C. Briand, "Adaptive random testing: An illusion of effectiveness?" in *ISSTA*, 2011.
19. A. Gargantini and C. Heitmeyer, "Using model checking to generate tests from requirements specifications," *Software Engineering Notes*, vol. 24, no. 6, pp. 146–162, November 1999.
20. R. Majumdar and K. Sen, "Hybrid concolic testing," in *ICSE*, 2007, pp. 416–426.
21. Y. Yu and M. Lau, "A comparison of MC/DC, MUMCUT and several other coverage criteria for logical decisions," *Journal of Systems and Software*, vol. 79, no. 5, pp. 577–590, 2006.
22. S. Kandl and R. Kirner, "Error detection rate of MC/DC for a case study from the automotive domain," *Software Technologies for Embedded and Ubiquitous Systems*, pp. 131–142, 2011.
23. A. Dupuy and N. Leveson, "An empirical evaluation of the MC/DC coverage criterion on the hete-2 satellite software," in *Proc. of the Digital Aviation Systems Conference (DASC)*, Philadelphia, USA, October 2000.
24. "Reactive systems inc. Reactis Product Description," http://www.reactive-systems.com/index.msp.
25. "Mathworks Inc. Simulink product web site," http://www.mathworks.com/products/simulink.
26. "Mathworks Inc. Stateflow product web site," http://www.mathworks.com.
27. N. Halbwachs, *Synchronous Programming of Reactive Systems*. Klower Academic Press, 1993.
28. A. Rajan, M. Whalen, M. Staats, and M. Heimdahl, "Requirements coverage as an adequacy measure for conformance testing," in *Proc. of the 10th Int'l Conf. on Formal Methods and Software Engineering*. Springer, 2008, pp. 86–104.
29. J. Andrews, L. Briand, and Y. Labiche, "Is mutation an appropriate tool for testing experiments?" *Proc of the 27th Int'l Conf on Software Engineering (ICSE)*, pp. 402–411, 2005.
30. C. Van Eijk, "Sequential equivalence checking based on structural similarities," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Trans. on*, vol. 19, no. 7, pp. 814–819, 2002.
31. J. Chilenski, "An investigation of three forms of the modified condition decision coverage (MCDC) criterion," Office of Aviation Research, Washington, D.C., Tech. Rep. DOT/FAA/AR-01/18, April 2001.
32. "The NuSMV Toolset," 2005, available at http://nusmv.irst.itc.it/.
33. R. Fisher, *The Design of Experiment*. New York: Hafner, 1935.
34. G. Devaraj, M. Heimdahl, and D. Liang, "Coverage-directed test generation with model checkers: Challenges and opportunities," *Computer Software and Applications Conference, Annual International*, vol. 1, pp. 455–462, 2005.
35. 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*. ACM New York, NY, USA, 2008, pp. 161–170.