Requirements Coverage as an Adequacy Measure for Conformance Testing*

Ajitha Rajan[1], Michael Whalen[2], Matt Staats[1], and Mats P.E. Heimdahl[1]

[1] University of Minnesota
arajan@cs.umn.edu, staats@cs.umn.edu, heimdahl@cs.umn.edu
[2] Rockwell Collins Inc.
mwwhalen@rockwellcollins.com

Abstract. Conformance testing in model-based development refers to the testing activity that verifies whether the code generated (manually or automatically) from the model is behaviorally equivalent to the model. Presently the adequacy of conformance testing is inferred by measuring structural coverage achieved over the model. We hypothesize that adequacy metrics for conformance testing should consider *structural coverage over the requirements* either in place of or in addition to structural coverage over the model. Measuring structural coverage over the requirements gives a notion of how well the conformance tests exercise the required behavior of the system.

We conducted an experiment to investigate the hypothesis stating structural coverage over formal requirements is more effective than structural coverage over the model as an adequacy measure for conformance testing. We found that the hypothesis was rejected at 5% statistical significance on three of the four case examples in our experiment. Nevertheless, we found that the tests providing requirements coverage found several faults that remained undetected by tests providing model coverage. We thus formed a second hypothesis stating that complementing model coverage with requirements coverage will prove more effective as an adequacy measure than solely using model coverage for conformance testing. In our experiment, we found test suites providing both requirements coverage and model coverage to be more effective at finding faults than test suites providing model coverage alone, at 5% statistical significance. Based on our results, we believe existing adequacy measures for conformance testing that only consider model coverage can be strengthened by combining them with rigorous requirements coverage metrics.

1 Introduction

In critical avionics applications, the validation and verification phase (V&V) is particularly costly and consumes a disproportionably large share of the development resources. Thus, if the process of deriving test cases for V&V can be

* This work has been partially supported by NASA Ames Research Center Cooperative Agreement NNA06CB21A, NASA IV&V Facility Contract NNG-05CB16C, and the L-3 Titan Group.

© Springer-Verlag Berlin Heidelberg 2008

automated to provide test suites that satisfy the most stringent standards (such as DO-178B in civil avionics [20]), dramatic time and cost savings can be realized. The current trend towards model-based development is one attempt to address this problem. In model-based software development, the traditional testing process is split into two distinct activities: one activity that tests the model to *validate* that it accurately captures the customers' high-level requirements, and another testing activity that *verifies* whether the code generated (manually or automatically) from the model is behaviorally equivalent to (or conforms to) the model. (Note that by "model", we are referring specifically to a high level formal model written in a language such as Simulink or Lustre. Throughout this paper, we refer to this simply as a "model".) In this paper, we focus on the second testing activity—verification through conformance testing. There are currently several tools, such as model checkers, that provide the capability to automatically generate conformance tests [19, 7] from formal models. In this paper, we examine the effectiveness of metrics used in measuring the adequacy of the generated conformance tests.

For critical avionics software, DO-178B necessitates test cases used in verification to achieve requirements coverage in addition to structural coverage over the code. However, there is no direct and objective measure of requirements coverage, and adequacy of tests is instead inferred by examining structural coverage achieved over the model. The Modified Condition and Decision Coverage (MC/DC) used when testing highly critical software [20] in the avionics industry has been a natural choice to measure structural coverage for the most critical models. In our work [21], however, we have defined coverage metrics that provide *direct and objective* measures of how well a test suite exercises a set of high-level formal software requirements. We examined using requirements coverage metrics, in particular the Unique First Cause (UFC) coverage metric, to measure adequacy of tests used in model validation (or black-box testing) and found them to be useful. To save time and effort, we would like to re-use validation tests providing requirements coverage for verification of code through conformance testing as well. This paper examines the suitability of using tests providing requirements UFC coverage for conformance testing as opposed to tests providing MC/DC over the model.

We believe requirements coverage will be useful as an adequacy measure for conformance testing for several reasons. First, measuring structural coverage over the requirements gives a direct assessment of how well the conformance tests exercise the required behavior of the system. Second, if a model is missing functionality, measuring structural coverage over the model will not expose such defects of omission. Third, obligations for requirements coverage describe satisfying scenarios (paths) in the model as opposed to satisfying states defined by common model coverage obligations (such as MC/DC). We believe coverage obligations that define satisfying paths will necessitate longer and more effective test cases than those defining satisfying states in the model. Finally, we found in [16] that structural coverage metrics over the model, in particular MC/DC, are sensitive to the structure of the model used in coverage measurement. Therefore, these metrics can be easily rendered inefficient by (purposely or inadvertently) restructuring the model to make it easier to achieve the desired coverage.

For these reasons, we believe that requirements coverage will serve as a stronger adequacy measure than model coverage in measuring adequacy of conformance test suites. More specifically, we investigate the following hypothesis in this paper:

Hypothesis 1 (H_1). *Conformance tests providing requirements UFC coverage are more effective at fault finding than conformance tests providing MC/DC over the model.*

We evaluated this hypothesis on four industrial examples from the civil avionics domain. The requirements for these systems are formalized as Linear Temporal Logic (LTL) [5] properties. The systems were modeled in the Simulink notation [12]. Using the Simulink models, we created implementations that we used as the basis for the generation of large sets of mutants by randomly seeding faults. We generate numerous test suites to provide 100% achievable UFC coverage over the LTL properties (the formal requirements), and numerous test suites to provide 100% achievable MC/DC over the model. We assessed the effectiveness of the different test suites by measuring their fault finding capability, i.e., running them over the sets of mutants and measuring the number of faults detected.

In our experiment we found that Hypothesis 1 was rejected on three of the four examples at the 5% statistical significance level. This result was somewhat disappointing since we believed that the requirements coverage would be effective as a conformance testing measure. The astute reader might point out that the result might not be surprising since the effectiveness of the requirements-based tests providing UFC coverage heavily depends on the 'goodness' of the requirements set; in other words, a poor set of requirements leads to poor tests. In this case, however, we worked with case examples with very good sets of requirements and we had expected better results. Nevertheless, we found that the tests providing requirements UFC coverage found several faults that remained undetected by tests providing MC/DC over the model. We thus formed a second hypothesis stating that complementing model coverage with requirements coverage will prove more effective as an adequacy measure than solely using model coverage for conformance testing. To investigate this, we formulated and tested the following hypothesis:

Hypothesis 2 (H_2). *Conformance tests providing requirements UFC coverage in addition to MC/DC over the model are more effective at fault finding than conformance tests providing only MC/DC over the model.*

In our second set of experiments, the combined test suites were significantly more effective than MC/DC test suites on three of the four case examples (at the 5% statistical significance level). For these examples, UFC suites found several faults not revealed by the MC/DC suites making the combination of UFC and MC/DC more effective than MC/DC alone. The relative improvement was in the range of 4.3% – 10.8% on these examples. We strongly believe that for the case example that did not support Hypothesis 2, the MC/DC suite found all possible faults, making improvement with the combined suites impossible. Based on our results, we believe that existing adequacy measures for conformance testing based solely on structural coverage over the model (such as MC/DC) can be strengthened by combining them with requirements coverage metrics such as UFC. It is worth noting that Briand et.al. found similar results in their study [3], though in the context of state-based testing for complex component models in object-oriented software. Combining a state-based testing technique for classes or class clusters modeled with statecharts [8], with a black-box testing technique, category partition testing, proved significantly more effective in fault detection. We recommend future measures of conformance testing adequacy to consider both requirements and model coverage either by combining existing metrics, such as MC/DC and UFC, or by defining new metrics that account for both.

The remainder of the paper is organized as follows. Section 2 introduces our experimental setup and the case examples used in our investigation. Results and statistical analysis are presented in Section 3. Finally in Sections 4 and 5, we analyze and discuss the implications of our results, and point to future directions.

2 Experiment

We use four industrial systems in our experiment: two models from a display window manager for an air-transport class aircraft (DWM_1, DWM_2), and two models representing flight guidance mode logic for business and regional jet class aircrafts (Vertmax_Batch and Latctl_Batch). All four systems were viewed to have good sets of requirements as judged by the developer of the system. We conducted the experiments for each case example using the steps outlined below (elaborated in later sections):

1. Generate and reduce test suites to provide requirements UFC coverage: We generated a test suite to provide UFC coverage over the formalized LTL requirements. This test suite was naïvely generated, one test case for every UFC obligation, and thus highly redundant. We reduced the test suite randomly while maintaining UFC coverage over the requirements. We generated three such randomly reduced test suites.

2. Generate and reduce test suites to provide MC/DC over the model: We naïvely generated a test suite to provide MC/DC over the model. We then randomly reduced the test suite to maintain MC/DC over the model. We generated three such reduced test suites.

3. Combined test suites that provide MC/DC + requirements UFC: Among the reduced MC/DC suites from the previous step, we selected the most effective MC/DC test suite based on their fault finding ability. We merge this test suite with each of the reduced UFC test suites from the first step. The combined suites thus provide both MC/DC over the model and UFC coverage over the requirements.

4. Generate mutants: We randomly seeded faults in the correct implementation and generated three sets of 200 mutants using the method outlined in Section 2.3.

5. Assess and compare fault finding: We run each of the test suites from steps 1, 2 and 3 (that provide requirements UFC coverage, MC/DC over the model, and MC/DC + requirements UFC coverage respectively) against each set of mutants and the model. Note that the model serves as the oracle implementation in conformance testing. We say that a mutant is killed (or detected) by a test suite if any of the test cases in the suite results in different output values between the model and the mutant. We recorded the number of mutants killed by each test suite and computed the fault finding ability as the percentage of mutants killed to the total number of mutants seeded.

2.1 Case Examples

In our experiment, we use four industrial systems. All four systems were modeled using the Simulink notation from Mathworks Inc.

Display Window Manager Models (DWM_1 and DWM_2): The Display Window Manager models, DWM_1, and DWM_2, represent 2 of the 5 major subsystems of the Display Window Manager (DWM) of an air transport-level commercial displays system. The DWM acts as a 'switchboard' for the system and has several responsibilities related to routing information to the displays and manages the location of two cursors that can be used to control applications by the pilot and copilot.

Flight Guidance System: A Flight Guidance System is a component of the overall Flight Control System (FCS) in a commercial aircraft. It compares the measured state of an aircraft (position, speed, and altitude) to the desired state and generates pitch and roll-guidance commands to minimize the difference between the measured and desired state. The FGS consists of the mode logic, which determines which lateral and vertical modes of operation are active and armed at any given time, and the flight control laws that accept information about the aircraft's current and desired state and compute the pitch and roll guidance commands. The two FGS models in this paper focus on the mode logic of the FGS. The Vertmax_Batch and Latctl_Batch models describe the vertical and lateral mode logic for the flight guidance system.

2.2 Test Suite Generation and Reduction

We generated test suites to provide UFC coverage over formal LTL requirements and to provide MC/DC over the model. The approach to generate and reduce the test suites for the two different coverage measures is detailed below. Additionally, we merge the reduced test suites for the two coverage measures to create combined test suites that provide UFC coverage over the requirements in addition to MC/DC over the model.

UFC Coverage over Requirements: The requirements coverage metric used in this paper is the *Unique First Cause* (UFC) coverage defined in [21].

The UFC metric is adapted from the Modified Condition/Decision Coverage (MC/DC) criterion [4, 9] defined over source code that defines satisfying states in the implementation. Since requirements captured as LTL properties define paths rather than states, we broadened our view of structural coverage to accommodate satisfying paths rather than satisfying states. We defined these satisfying test paths by extending the constraints for state-based MC/DC to include temporal operators. A test suite is said to satisfy UFC coverage over a set of LTL formulae if executing the test cases in the test suite will guarantee that:

– Every basic condition in a formula has taken on all possible outcomes at least once
– Each basic condition has been shown to independently affect the formula's outcome.

We defined independence in terms of the shortest satisfying path for the formula. Thus, if we have a formula A and a path π, an atomic condition α in A is the unique first cause if, in the first state along π in which A is satisfied, it is satisfied because of atomic condition α. The formal definition for UFC and the obligations for LTL temporal operators was presented in [21].

The notion of requirements UFC coverage used in this paper is related to work assessing the completeness and correctness of formulae in temporal logics, in particular, *vacuity checking* of temporal logic formulas [2, 10, 15]. The focus of this paper, however, is on the application and usefulness of requirements coverage to measure adequacy of conformance testing, and not on the completeness of requirements. We are not aware of studies that investigated applicability of requirements coverage in this context.

Several research efforts have developed techniques for automatic generation of tests from formal models using model checkers as test case generation tools [17, 18, 7]. One such technique operates by formulating a test criterion as a verification condition for the model checker. The obligations for requirements UFC coverage are given as trap properties (by negating the obligations) to the model checker along with the formal model of the system, and the model checker returns counter examples that constitute a test suite for UFC coverage over the LTL requirements.

A test suite thus generated will be highly redundant, as a single test case will often satisfy several UFC obligations. We therefore reduce this test suite using a greedy approach. We randomly select a test case from the test suite, check how many UFC obligations are satisfied and add it to a reduced test set. Next, we randomly pick another test case from the suite and check whether any additional UFC obligations were satisfied. If so, we add the test case to the reduced test set. This process continues till we have exhausted all the test cases in the test suite. We now have a randomly reduced test suite that maintains UFC coverage over the LTL requirements. We generate three such reduced UFC test suites for each case example in our experiment to eliminate the possibility of skewing our results with an outlier (an extremely good or bad reduced test suite).

MC/DC over model: The full test suite to provide MC/DC used in this experiment is the same one used in previous work [16]. We used the test suite that provides MC/DC over the inlined model rather than the non-inlined model as it is more rigorous and effective. We thus compare the requirements UFC coverage against a rigorous notion of MC/DC. The test suite was automatically generated using the NuSMV [13] model checker to provide MC/DC over the model. The full test suite was naïvely generated, with a separate test case for every construct we need to cover in the model. This straightforward method of generation results in highly redundant test suites, as with UFC test suite generation. Thus, the size of the complete test suite can typically be reduced while preserving coverage.

The approach to reduce the test suite is similar to that used for UFC coverage. As before, we generate three such reduced test suites to decrease the chances of skewing our results with an outlier (very good or very bad reduced test suite).

Requirements UFC Coverage + MC/DC over model: To generate test suites providing both requirements UFC coverage and MC/DC over the model, we simply merge the test suite providing UFC with the test suite providing MC/DC. As mentioned previously, we generated three reduced MC/DC suites and three reduced UFC suites. It is thus possible to create nine different combined suites, by merging each of the three reduced MC/DC suites with each of the three reduced UFC suites. Using all nine suites in our experiment, however, would have been very time consuming. To reduce the combinations, we instead elected to use only the best reduced MC/DC suite (with respect to fault finding among the reduced MC/DC suites) for creating the combined test suites. We thus merge the best MC/DC suite with each of the three reduced UFC suites to create only three combined suites. Note that choosing the best MC/DC implies that the combined suites must improve the fault finding of the best MC/DC suite to support Hypothesis 2.

2.3 Mutant Generation

To create mutants or faulty implementations, we built a fault seeding tool that can randomly inject faults into the implementation. Each mutant is created by introducing a single fault into a correct implementation by mutating an operator or variable.

The fault seeding tool is capable of seeding faults from different classes. We seeded the following classes of faults:

Arithmetic: Changes an arithmetic operator (+, -, /, *, mod, exp).
Relational: Changes a relational operator ($=, \neq, <, >, \leq, \geq$).
Boolean: Changes a boolean operator ($\vee, \wedge$, XOR).
Negation: Introduces the boolean $\neg$ operator.
Delay: Introduces the delay operator on a variable reference (that is, use the stored value of the variable from the previous computational cycle rather than the newly computed value).

Constant: Changes a constant expression by adding or subtracting 1 from int and real constants, or by negating boolean constants.

Variable Replacement: Substitutes a variable occurring in an equation with another variable of the same type.

To seed a fault from a certain class, the tool first randomly picks one expression among all possible expressions of that kind in the implementation. It then randomly determines how to change the operator. For instance to seed an arithmetic mutation, we first randomly pick one expression from all possible arithmetic expressions to mutate, say we pick the expression 'a + b'; we then randomly determine if the arithmetic operator '+' should be replaced with '-' or '*' or '/' and create the arithmetic mutant accordingly. Our fault seeding tool ensures that no duplicate faults are seeded.

In our experiment, we generated mutants so that the 'fault ratio' for each fault class is uniform. The term fault ratio refers to the number of mutants generated for a specific fault class versus the total number of mutants possible for that fault class. For example, assume an implementation consists of R Relational operators and B Boolean operators. Thus there are R possible Relational faults and B possible Boolean faults. For uniform fault ratio, we would seed x relational faults and y boolean faults in the implementation so that $x/R = y/B$.

We generated three sets of 200 mutants for each case example. We generated multiple mutant sets for each example to reduce potential bias in our results from a mutant set that may have very hard (or easy) faults to detect. Our mutant generator does not guarantee that a mutant will be semantically different from the original implementation. Nevertheless, this weakness in mutant generation does not affect our results, since we are investigating the relative fault finding of test suites rather than the absolute fault finding.

The fault finding effectiveness of a test suite is measured as the number of mutants detected (or 'killed') to the total number of mutants created. We say that a mutant is detected by a test suite when the test suite results in different observed values between the mutant and the oracle implementation. The system model serves as the oracle implementation in conformance testing. We only observe the output values of the model and mutants for comparison. We do not use internal state information, as internal state information between the model and implementation may differ and can therefore not be compared directly. Additionally, internal state information of the system under test may not be available during real-world tests and it is therefore preferable to perform the comparison with only output values.

3 Experimental Results

For each case example described in Section 2.1, we generated three reduced UFC test suites, three reduced MC/DC test suites, three combined UFC + MC/DC test suites and three sets of mutants. As mentioned earlier the combined suites are created by merging the best reduced MC/DC suite with each of the three reduced UFC suites. For this reason we compare the fault finding ability of the combined suites only against the best MC/DC suite rather than all the reduced MC/DC suites. We ran every test suite against every set of mutants, and recorded the percentage of mutants caught. For each case example, this yielded nine observations each for MC/DC, UFC and the combined test suites. We average the percentage of mutants caught across the mutant sets for each case example and each kind of test suite. This yields three averages, one each for MC/DC, UFC, and combined test suites as summarized in Table 1. Also, for each case example we identify the most effective MC/DC suite (among the generated three reduced MC/DC suites) and calculate average fault finding across the mutant sets. The 'Best MC/DC' column in the table represents these averaged observations. Table 1 also gives the relative improvement in average fault finding of UFC suites over MC/DC test suites, and combined suites over the best MC/DC suite. Note that some of the numbers in the relative improvement column in Table 1 are negative. This implies that the test suite did not yield an improvement, and instead did worse than the MC/DC test suite at fault finding. For instance, for the DWM_1 model the MC/DC test suites provide an average fault finding of 84.6% and the UFC suites provide an average fault finding of 82.7%, and thus the relative improvement in fault finding for UFC suites is negative (= -2.2%) with respect to MC/DC suites. Conversely, for the DWM_1 system, the combined suites provide better fault finding (an average of 91.5%) than the best MC/DC suite (85.8%), giving a positive relative improvement of 6.6%.

The complete set of 27 fault finding observations for each case example is presented in Table 2. Note that in the table, $M1$, $M2$, $M3$ denote the three mutant sets; $MCDC\_1$, $MCDC\_2$, $MCDC\_3$ refer to the reduced MC/DC suites; UFC_1, UFC_2, UFC_3 to reduced UFC suites; and C_1, C_2, C_3 to the combined UFC and MC/DC suites. The best MC/DC suite (used to create combined suites) can be identified by comparing the fault finding of $MCDC\_1$, $MCDC\_2$, and $MCDC\_3$ across the mutant sets. Thus from the results in Table 2 we find for the DWM_1 system, the best MC/DC test suite is $MCDC\_3$. For DWM_2 and Vertmax_Batch systems, all three reduced MC/DC suites are equally effective so any of them can be used for creating the combined suites. We randomly selected $MCDC\_1$ for DWM_2 and $MCDC\_3$ for Vertmax_Batch system. Finally, for the Latctl_Batch system, $MCDC\_1$ is the most effective.

From the results in Tables 1 and 2, it is evident that for all case examples, except the Latctl_Batch system, MC/DC test suites outperform the UFC suites

Table 1. Average percentage of mutants caught by test suites and relative improvement over MC/DC

	Avg. MC/DC	**Avg. UFC**	**Relative Improv.**	**Best MC/DC**	**Avg. Combined**	**Relative Improv.**
DWM_1	84.6%	82.7%	-2.2%	85.8%	91.5%	6.6 %
DWM_2	90.6%	16.7%	-81.6%	90.6%	90.6%	0.0%
Latctl_Batch	85.1%	88.7%	4.2%	85.4%	94.6%	10.8%
Vertmax_Batch	86.0%	68.6%	-20.2%	86.0%	89.7%	4.3%

Table 2. Complete results for all case examples

	DWM_1								
	MCDC_1	**MCDC_2**	**MCDC_3**	**UFC_1**	**UFC_2**	**UFC_3**	**C_1**	**C_2**	**C_3**
M1	82.7%	81.2%	84.3%	79.2%	79.7%	76.1%	88.3%	89.3%	88.8%
M2	83.8%	83.8%	86.3%	83.8%	82.7%	81.2%	91.4%	91.9%	91.4%
M3	86.3%	86.3%	86.8%	87.3%	87.8%	86.8%	93.9%	94.4%	94.4%
TS Size	73	76	77	463	469	468	540	546	545
	DWM_2								
	MCDC_1	**MCDC_2**	**MCDC_3**	**UFC_1**	**UFC_2**	**UFC_3**	**C_1**	**C_2**	**C_3**
M1	91.4%	91.4%	91.4%	17.2%	15.2%	16.7%	91.4%	91.4%	91.4%
M2	91.4%	91.4%	91.4%	16.7%	14.6%	16.7%	91.4%	91.4%	91.4%
M3	88.9%	88.9%	88.9%	18.7%	16.2%	18.7%	88.9%	88.9%	88.9%
TS Size	452	452	448	33	32	31	485	484	483
	Latctl_Batch								
	MCDC_1	**MCDC_2**	**MCDC_3**	**UFC_1**	**UFC_2**	**UFC_3**	**C_1**	**C_2**	**C_3**
M1	85.2%	84.2%	84.7%	89.3%	87.8%	89.8%	94.4%	92.9%	91.8%
M2	85.7%	85.7%	85.2%	89.3%	89.3%	89.8%	96.4%	95.9%	95.9%
M3	85.2%	84.7%	85.2%	88.8%	85.2%	88.8%	94.9%	93.9%	94.9%
TS Size	73	71	73	50	49	53	123	122	126
	Vertmax_Batch								
	MCDC_1	**MCDC_2**	**MCDC_3**	**UFC_1**	**UFC_2**	**UFC_3**	**C_1**	**C_2**	**C_3**
M1	83.8%	83.8%	83.8%	67.5%	69.5%	66.0%	88.8%	88.8%	88.8%
M2	81.3%	81.3%	81.3%	71.1%	71.1%	71.6%	91.9%	90.4%	90.4%
M3	88.9%	88.9%	88.9%	64.5%	66.0%	70.1%	87.3%	86.3%	88.3%
TS Size	301	299	297	89	79	88	386	376	385

in fault finding. The degree to which MC/DC suites are better, however, varies by a vast range. The maximum difference is on DWM_2, where MC/DC suites provide an average fault finding of 90.6% in contrast to 16.7% provided by UFC suites. The minimum difference is on DWM_1 where MC/DC provides an average fault finding of 84.6% versus 82.7% provided by UFC suites. The combined suites on the other hand outperform the MC/DC suites. The relative improvement provided by the combined suites however spans a much smaller range (0 - 10.8%). In other words, the number of different faults revealed by the UFC suites as compared to the best MC/DC suite is in the range of $0 - 10.8\%$ of the mutants seeded. The combined suites provide better fault finding than the best MC/DC suite on three of the four case examples. On the DWM_2 system the combined suites yield no improvement. A detailed discussion of the implication of these results is presented in Section 4.

3.1 Statistical Analyses

In this section, we statistically analyze the results in Tables 1 and 2 to determine if the hypotheses H_1 and H_2, stated previously in Section 1, are supported.

To evaluate H_1 and H_2, we formulate our respective null hypotheses $H0_1$ and $H0_2$ as follows:

$H0_1$: A test suite generated to provide requirements UFC coverage will find the same number of faults as a test suite generated to provide MC/DC coverage over the model.

$H0_2$: A test suite generated to provide both requirements UFC coverage and MC/DC over the model will reveal the same number of faults as a test suite generated to provide only MC/DC over the model.

To accept H_2, we must reject $H0_2$. Rejecting $H0_2$ implies that the data for the combined test suite and MC/DC suite come from different populations. In other words, this implies that either the combined suites have more fault finding than the MC/DC suites or vice versa. However, the combined suite includes the MC/DC suite and can therefore never have lesser fault finding than the MC/DC suite. This implies that H_2 is supported when $H0_2$ is rejected. On the other hand, rejecting $H0_1$ does not necessarily imply H_1 is supported, as this implies that the UFC suites have *different* fault finding ability than the MC/DC suites, not necessarily better fault finding ability. To accept H_1 after rejecting $H0_1$, we examine the data in the table and determine if the UFC suites have greater fault finding than the MC/DC suite. If so, we accept H_1. If the data indicates that UFC suites instead have lesser fault finding than the MC/DC suites, we reject H_1.

Our experimental observations are drawn from an unknown distribution, and we therefore cannot reasonably 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 permutation test, a non-parametric test with no distribution assumptions. When performing a permutation test, a reference distribution is obtained by calculating all possible permutations of the observations [6, 11]. To perform the permutation test, we restate our null hypotheses as:

$H0_1$: The data points for percentage of mutants caught using the UFC and MC/DC test suites come from the same population.

$H0_2$: The data points for percentage of mutants caught using the MC/DC and combined UFC + MC/DC test suites come from the same population.

We evaluate the two hypotheses for each of the case examples. The procedure for permutation test of each hypothesis is as follows. Data is partitioned into two groups: A and B. Null hypothesis states that data in groups A and B come from the same population. We calculate the test statistic S as the absolute value of the difference in the means of group A and B:

$$S = \mid \overline{A} - \overline{B} \mid$$

We calculate $Number\ of\ Permutations$ as the number of ways of grouping all the observations in A and B into two sets. We then let $COUNT$ equal the number of permutations of A and B in which the test statistic is greater than S. Finally, $P - Value$ is calculated as:

$$P - Value = COUNT\ /\ Number\ of\ Permutations$$

For each case example, if $P - Value$ is less than the α value of 0.05 then we reject the null hypothesis with significance level α.

The null hypotheses $H0_1$ and $H0_2$ are evaluated using different groups of data. For $H0_1$, data for each case example in Table 2 is partitioned into two groups with nine observations each: % of faults caught by UFC test suites (group A – columns UFC_1, UFC_2, UFC_3 in the table), and % of faults caught by MC/DC test suites (group B – columns $MCDC\_1$, $MCDC\_2$, $MCDC\_3$). We calculate the $Number\ of\ Permutations$ as:

$$Number\ of\ Permutations = \binom{18}{9} = 48620$$

For $H0_2$, data for each case example in Table 2 is partitioned into two groups, one with nine observations and the other with three observations: % of faults caught by combined UFC+MC/DC test suites (group A – columns C_1, C_2, C_3), and % of faults caught by the best MC/DC suite (group B – $MCDC\_1$ column for DWM_2 and Latctl_Batch systems, and $MCDC\_3$ column for DWM_1 and Vertmax_Batch systems). We calculate the $Number\ of\ Permutations$ as:

$$Number\ of\ Permutations = \binom{12}{9} = 220$$

We then determine the p-value for each hypothesis using the procedure described previously. Table 3 lists the p-values for both null hypotheses ($H0_1$ and $H0_2$) and states if the corresponding original hypotheses (H_1 and H_2) are supported for each case example. As mentioned earlier, for each case example, H_1 is supported if $H0_1$ is rejected with significance level $\alpha = 0.05$ and all the UFC suites (columns UFC_1, UFC_2, UFC_3 in Table 2) have better fault finding than the MCDC suites (columns $MCDC\_1$, $MCDC\_2$, $MCDC\_3$), and H_2 is supported if we reject $H0_2$.

Given the p-values in Table 3 and the fault finding data in Table 2 we examine why the original hypotheses (H_1 and H_2) are supported/rejected for each case example. For the DWM_1 system, $H0_1$ is accepted (since p-value is greater than α value), and we therefore reject H_1. For the other three systems, $H0_1$ is rejected but the UFC suites outperform the MC/DC suites only on the Latctl_Batch system. For the DWM_2 and Vertmax_Batch systems, MC/DC suites always outperform the UFC test suites. Thus, H_1 is supported on the Latctl_Batch system and rejected on the DWM_2 and Vertmax_Batch systems. On the other hand, $H0_2$ is rejected (p-value less than the α value) on all but the DWM_2 system. This implies that H_2 is supported on all except the DWM_2 system. Thus, we find that

Table 3. Hypotheses Evaluation for different case examples

	P-Value		**Result**	
	$H0_1$	$H0_2$	H_1	H_2
DWM_1	0.24	0.004	Unsupported	Supported
DWM_2	0.00004	1.0	Unsupported	Unsupported
Latctl_Batch	0.0002	0.004	Supported	Supported
Vertmax_Batch	0.00004	0.027	Unsupported	Supported

with statistical significance level $\alpha = 0.05$ hypothesis H_1 is supported only on one case example, and hypothesis H_2 is supported on three of the four case examples.

3.2 Threats to Validity

While our results are statistically significant, they are derived from a small set of examples, which poses a threat to the generalization of the results. Nevertheless, we believe that the examples in our experiment are highly representative and our results are generalizable to systems within the same domain.

Our fault seeding method seeds one fault per mutant. In practice, implementations are likely to have more than one fault. However, previous studies have shown that mutation testing in which one fault is seeded per mutant draws valid conclusions of fault finding ability [1].

Additionally, all fault seeding methods have an inherent weakness. It is difficult to determine the exact fault classes and ensure that seeded faults are representative of faults that occur in practical situations. In our experiment, we assume a uniform ratio of faults across fault classes. This may not reflect the fault distribution in practice. Finally, our fault seeding method does not ensure that seeded faults result in mutants that are semantically different from the oracle implementation. Ideally, we would eliminate mutants that are semantically equivalent, however, identifying such mutants is infeasible in practice.

4 Discussion

In this section we analyze and discuss the implications of the results in Tables 1 and 2. We present the discussion in the context of Hypotheses 1 and 2 stated in Section 1.

4.1 Analysis - Hypothesis 1

As seen from Table 1, on all but one of the industrial systems, test suites generated for requirements UFC coverage have lower fault finding than test suites providing MC/DC over the system model. Statistical analysis revealed that hypothesis 1 stating "test suites providing requirements UFC coverage have better fault finding than test suites providing MC/DC over the model" was supported only on the Latctl_Batch system and rejected on all the other systems at the 5% significance level. We believe this may be because of one or both of the following reasons, (1) The UFC metric used for requirements coverage is not sufficiently rigorous and we thus have an inadequate set of requirements-based tests, and (2) Requirements are not sufficiently defined with respect to the system model. Thus, test suites providing requirements coverage will be ineffective at revealing faults in the model since there are behaviors in the model not specified in the requirements.

To assess the rigor of the UFC metric and the quality of the requirements with regard to behaviors covered in the system model, we measured MC/DC achieved by the reduced UFC suites over the system model. The results are summarized in Table 4. We found that for all the case examples, UFC test suites provide less

Table 4. MC/DC achieved by the reduced UFC suites over the system model

	Avg. MC/DC Achieved by UFC suites	**Achievable MC/DC**	**Rel. Diff.**
DWM_1	78.2%	92.5%	15.5%
DWM_2	25.8%	100%	74.2%
Latctl_Batch	88.6%	98.0%	9.6%
Vertmax_Batch	80.9%	99.8%	18.9%

than Achievable MC/DC over the system model. Thus, faults seeded in these uncovered portions of the model cannot be revealed by the UFC suites. The extent to which the model is covered is an indicator of the effectiveness of the UFC metric and the quality of the requirements set. On the DWM_1, Vertmax_Batch, and Latctl_Batch systems the UFC suites do reasonably well, achieving an average MC/DC of 78.2%, 88.6%, and 80.9% respectively as compared to 92.5%, 98% and 99.8% achievable MC/DC. Note, however, that relative differences in MC/DC need not correspond exactly to relative differences in fault finding between the UFC and MC/DC suites (as seen in our examples). In addition to coverage, fault finding is also highly influenced by the nature and number of faults seeded in covered and uncovered portions of the model. The relation between coverage and fault finding is not the focus of this paper and we hope to investigate this in our future work.

On the DWM_2 system, the UFC suites do poorly in both fault finding and MC/DC achieved. The UFC suites only achieve an average of 25.8% MC/DC over the model when compared to an achievable MC/DC of 100%. Correspondingly, the UFC suites have very poor fault finding (average of 16.7%) when compared to the MC/DC suites (average of 90.6%), since faults seeded in the uncovered portions of the model cannot be revealed by the UFC suites. The terrible fault finding and MC/DC achieved by the UFC suites on the DWM_2 system was surprising since we knew the system had a good set of requirements. To gain better understanding we took a closer look at the requirements set and the UFC obligations generated from them. We found that many of the requirements were structured similar to the sample requirement (formalized as an LTL property in the SMV [13] language) below,

```
LTLSPEC G(var_a > (
    case
        foo : 0 ;
        bar : 1 ;
    esac +
    case
        baz : 2 ;
        bpr : 3 ;
    esac
));
```

Informally, the sample requirement states that *var_a* is always greater than the sum of the outcomes of the two case expressions. When we perform UFC for the above requirement, it would result in obligations for the following expressions:

1. Relational expression within the globally operator (G)
2. Atomic condition foo within the first case expression
3. Atomic condition *bar* within the first case expression
4. Atomic condition *baz* within the second case expression
5. Atomic condition *bpr* within the second case expression

The above requirement may be restructured (to express the same behavior) so that the sum of two case expressions is expressed as a single case expression as shown:

```
LTLSPEC G(var_a > (
    case
        foo & baz : 0 + 2 ;
        foo & bpr : 0 + 3 ;
        bar & baz : 1 + 2 ;
        bar & bpr : 1 + 3 ;
    esac
));
```

Achieving UFC coverage over this restructured requirement will involve more obligations than before since the boolean conditions in the case expression are more complex. UFC would result in obligations for the following expressions in this restructured requirement:

1. Relational expression within the globally operator (G)
2. Complex condition foo & *baz* within the case expression
3. Complex condition foo & *bpr* within the case expression
4. Complex condition *bar* & *baz* within the case expression
5. Complex condition *bar* & *bpr* within the case expression

Thus, the structure of the requirements has a significant impact on the number and rigor of UFC obligations and hence the size of the test suite providing UFC coverage. In our experiment, we did not restructure requirements similar to the sample requirement discussed and instead retained the original structure. Therefore, the UFC obligations generated were fewer and far less rigorous. We believe this is the primary reason for the poor performance (both fault finding and MC/DC achieved) of the UFC suites for the DWM_2 system. The experience with the DWM_2 system suggests that even with a good set of requirements, rigorous requirements coverage metrics, such as the UFC metric, can be easily cheated since they are highly sensitive to the structure of the requirements. The issue here is similar to the sensitivity of the MC/DC metric to structure of the implementation observed in [16]. MC/DC was found to be significantly less effective when used over an implementation structure with intermediate variables and non inlined function calls as opposed to an implementation with inline expanded intermediate variables and function calls. Thus, as with all structural coverage metrics, we must be aware that the *structure* of the object used in measurement plays an important role in the effectiveness of the metrics.

To summarize, we find that the fault finding effectiveness of test suites providing requirements UFC coverage is heavily dependent on the nature and completeness of the requirements. Additionally, the rigor and robustness (with respect to requirements structure) of the requirements coverage metric used plays an important role in the effectiveness of the generated test suites. Thus, even with a good set of requirements, test suites providing requirements structural coverage may be ineffective if the coverage metric can be cheated. In our experiment, the UFC metric gets cheated when requirements are structured to hide the complexity of conditions on the DWM_2 system. Based on these observations and our results, we do not recommend using requirements coverage in place of model coverage as a measure of adequacy for conformance test suites.

4.2 Analysis - Hypothesis 2

As seen in Tables 1 and 2, for three of the four industrial case examples the combined UFC and MC/DC suites outperform the MC/DC suite in fault finding. For the DWM_2 system, however, the combined suites yield no improvement in fault finding over the MC/DC suite. Statistical analysis on the data in Table 2 revealed that Hypothesis 2 is supported with a significance level of 5% for the DWM_1, Vertmax_Batch, Latctl_Batch systems, and rejected for the DWM_2 system since the combined suites yield no improvement.

For the the DWM_1, Vertmax_Batch, Latctl_Batch systems, the combined UFC and MC/DC suites yielded an average fault finding improvement in the range of (4.3% - 10.8%) over the best MC/DC suite. The relative improvement implies that the UFC suites find a considerable number of faults not revealed by the best MC/DC suite.

To confirm that the improvement seen in DWM_1, Vertmax_Batch, Latctl_Batch systems is a result of combining the MC/DC metric with the UFC metric and not solely because of the increased number of test cases in the combined suites, we decided to measure the UFC coverage achieved over the requirements by the MC/DC suite. The results are summarized in Table 5. To understand the implications of the results in the table, consider the following two situations. If the MC/DC suite provides 100% achievable UFC over the requirements, it implies that the combined MC/DC + UFC coverage is satisfied by simply using the MC/DC suite instead of the combined suites. Under such circumstances, the fault finding improvement observed on combining the test suites would be solely due to the increased number of test cases. On the

Table 5. UFC achieved by the reduced MC/DC suites over the system model

	Avg. UFC Achieved by MC/DC suites	**Achievable UFC**	**Rel. Diff.**
DWM_1	28.3%	96.9%	70.8%
DWM_2	59.7%	64.0%	6.7%
Latctl_Batch	94.7%	99.5%	4.8%
Vertmax_Batch	97.4%	99.0%	1.6%

other hand, if the MC/DC suite provides less than achievable UFC over the requirements, it implies that there are scenarios/behaviors specified by the requirements that are not covered by the MC/DC suite but covered by the UFC suite. Thus, the combination may have proved more effective because of these additional covered scenarios and not simply because of increased test cases. We now take a closer look at the results in Table 5 to see which of these situations occurred. We found that in all three systems (Latctl_Batch, Vertmax_Batch, and DWM_1), the MC/DC suites provided less than achievable UFC coverage over the requirements. This indicates that the UFC suites cover several behaviors specified in the requirements that are not covered by the MC/DC suite. We postulate that these additional covered behaviors contribute to the improved fault finding observed with the combined suites on these systems.

For the DWM_2 system, the combined suites yield no improvement in fault finding over the MC/DC suite, implying that the faults revealed by the UFC suites are a subset of the faults revealed by the MC/DC suite. The DWM_2 system consists almost entirely of complex Boolean mode logic, and the MC/DC metric is extremely effective for these type of systems. There is thus a distinct possibility that the MC/DC suite reveals all the seeded faults (excluding semantically equivalent faults that can never be revealed). This belief was strengthened when we ran the full rather than reduced MC/DC and UFC suites and measured fault finding. We found that even with the full test suites, which have a dramatically larger number of test cases, the combination did not yield any improvement in the number of faults revealed. Therefore, we believe that there is a strong possibility the MC/DC suites revealed all but the semantically equivalent faults on the DWM_2 system. Under such circumstances, no test suite complementing the MC/DC suite can improve the fault finding, thus forcing us to always reject Hypothesis 2. Such occurrences are anomalous and we discount them from our analysis.

To summarize, we found that for three of the four case examples, the combined test suite providing both requirements UFC coverage and MC/DC over the model is significantly more effective than a test suite solely providing MC/DC over the model. For the DWM_2 system that did not support this, we strongly believe that the MC/DC suites revealed all possible faults making improvement in fault finding on combining with UFC suites impossible. We disregard this abnormal occurrence to conclude that combined test suites have better fault finding than the MC/DC suites for all the systems. Given our results, we believe using requirements coverage metrics, such as UFC, in combination with model coverage metrics, such as MC/DC, yields a significantly stronger adequacy measure than simply covering the model.

Note that for all the case examples, all three kinds of test suites—MC/DC, UFC, and combined—never yield 100% fault finding. This is because some of the seeded faults may result in mutant implementations that are semantically equivalent to the correct implementation (i.e., faults that *cannot* result in any observable failure). This is a common problem in fault seeding experiments [1, 14]. In industrial size examples it is extraordinarily expensive and time consuming, or—in most cases—infeasible to identify mutations that are semantically equivalent to the correct implementation and exclude them from consideration. Therefore, the fault finding percentage that we give in our experiment results is a conservative estimate, and we expect the actual fault finding for the test suites to be higher if we were to exclude the semantically equivalent mutations. However, this issue will not affect our conclusions since we only judge based on relative fault finding rather than absolute fault finding.

5 Conclusions

Presently in model-based development, adequacy of conformance test suites is inferred by measuring structural coverage achieved over the model. In this paper we investigated the use of requirements coverage as an adequacy measure for conformance testing. Our empirical study revealed that on three of the four industrial case examples, our hypothesis stating "Requirements coverage (UFC) is more effective than model coverage (MC/DC) when used as an adequacy measure for conformance test suites" was rejected at 5% statistical significance level. Nevertheless, we found that requirements coverage is useful when used in combination with model coverage to measure adequacy of conformance test suites. Our hypothesis stating that "test suites providing both requirements UFC coverage and MC/DC over the model are more effective than test suites providing only MC/DC over the model" was supported at 5% significance level on three of the four case examples. The relative improvement yielded by the combined suites over the MC/DC suites was in the range of 4.3% – 10.8%. The system that did not support the hypothesis was an outlier where we firmly believe the MC/DC suite found all possible faults, making improvement with the combined suites impossible. Based on our results, we believe that the effectiveness of adequacy measures based solely on model coverage can certainly be improved. Combining existing metrics for model coverage and requirements coverage investigated in this paper may be one possible way of accomplishing this. There may be other approaches, for instance, defining a new metric that accounts for both requirements and model coverage. We hope to investigate this further in our future work.

Another observation gained in our experiment relates to the sensitivity of requirements coverage metrics such as UFC to the structure of the requirements. Test suites providing requirements coverage may be ineffective even with an excellent set of requirements. This can occur when structure of the formalized requirements effectively "cheats" the requirements coverage metric. The UFC metric in our experiment was cheated when requirements were structured to hide the complexity of conditions in them. In our future work, we hope to define requirements coverage metrics that are more robust to the structure of the requirements.

References

1. Andrews, J.H., Briand, L.C., Labiche, Y.: Is Mutation an Appropriate Tool for Testing Experiments? In: Proceedings of the 27th International Conference on Software Engineering (ICSE), pp. 402–411 (2005)

2. Beer, I., Ben-David, S., Eisner, C., Rodeh, Y.: Efficient detection of vacuity in ACTL formulas. In: Formal Methods in System Design, pp. 141–162 (2001)
3. Briand, L.C., Di Penta, M., Labiche, Y.: Assessing and Improving State-Based Class Testing: A Series of Experiments. IEEE Transactions on Software Engineering 30(11) (2004)
4. Chilenski, J.J., Miller, S.P.: Applicability of Modified Condition/Decision Coverage to Software Testing. Software Engineering Journal, 193–200 (September 1994)
5. Clarke, E.M., Grumberg, O., Peled, D.: Model Checking. MIT Press, Cambridge (1999)
6. Fisher, R.A.: The Design of Experiment. Hafner, New York (1935)
7. Gargantini, A., Heitmeyer, C.: Using model checking to generate tests from requirements specifications. Software Engineering Notes 24(6), 146–162 (1999)
8. Harel, D., Marelly, R.: Come Let's Play: Scenario-Based Programming Using LSC's and the Play-Engine. Springer, New York (2003)
9. Hayhurst, K.J., Veerhusen, D.S., Rierson, L.K.: A practical tutorial on modified condition/decision coverage. Technical Report TM-2001-210876, NASA (2001)
10. Kupferman, O., Vardi, M.Y.: Vacuity detection in temporal model checking. Journal on Software Tools for Technology Transfer 4(2) (February 2003)
11. Kvam, P.H., Vidakovic, B.: Nonparametric Statistics with Applications to Science and Engineering (2007)
12. Mathworks Inc. Simulink product web site, `http://www.mathworks.com/products/simulink`
13. The NuSMV Toolset (2005), `http://nusmv.irst.itc.it/`
14. Offutt, A.J., Pan, J.: Automatically detecting equivalent mutants and infeasible paths. Software Testing, Verification & Reliability 7(3), 165–192 (1997)
15. Purandare, M., Somenzi, F.: Vacuum cleaning CTL formulae. In: Proceedings of the 14th Conference on Computer Aided Design, pp. 485–499. Springer, Heidelberg (2002)
16. Rajan, A., Whalen, M.W., Heimdahl, M.P.E.: The Effect of Program and Model Structure on MC/DC Test Adequacy Coverage. In: Proceedings of 30th International Conference on Software Engineering (ICSE) (to appear, 2008), `http://crisys.cs.umn.edu/ICSE08.pdf`
17. Rayadurgam, S.: Automatic Test-case Generation from Formal Models of Software. PhD thesis, University of Minnesota (November 2003)
18. Rayadurgam, S., Heimdahl, M.P.E.: Coverage based test-case generation using model checkers. In: Proceedings of the 8th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS 2001), April 2001, pp. 83–91. IEEE Computer Society Press, Los Alamitos (2001)
19. Rayadurgam, S., Heimdahl, M.P.E.: Generating MC/DC adequate test sequences through model checking. In: Proceedings of the 28th Annual IEEE/NASA Software Engineering Workshop – SEW 2003, Greenbelt, Maryland (December 2003)
20. RTCA. DO-178B: Software Consideration. In: Airborne Systems and Equipment Certification. RTCA (1992)
21. Whalen, M.W., Rajan, A., Heimdahl, M.P.E.: Coverage metrics for requirements-based testing. In: Proceedings of International Symposium on Software Testing and Analysis (July 2006)