A Method for Analyzing System State-space Coverage within a t-Wise Testing Framework

Joshua R. Maximoff
Embedded Applications Group
JHU/APL
Laurel, MD
joshua.maximoff@jhuapl.edu

Michael D. Trela
Systems Engineering Group
JHU/APL
Laurel, MD
michael.trela@jhuapl.edu

D. Richard Kuhn
Computer Security Division
NIST
Gaithersburg, MD
d.kuhn@nist.gov

Raghu Kacker
Mathematical and Computation
Sciences Division
NIST
Gaithersburg, MD
raghu.kacker@nist.gov

***Abstract*— Inadequate state-space coverage of complex configurable systems during test phases is an area of concern for systems engineers. Determining the state-space coverage of a proposed or executed test suite traditionally involves qualitative assessment, rendering meaningful comparative analysis between tests for a given system or across multiple systems difficult. We propose a method for assessing state-space coverage of a test suite utilizing *t*-wise testing, a combinatorial technique borrowed from the software testing community which generalizes pair-wise testing. We refine traditional notions of a *t*-wise test suite to analyze the configuration coverage of a test plan. This provides a methodology and a set of metrics to assess both the level and the distribution of state-space coverage. We detail a proof-of-concept experiment using this partial *t*-wise coverage framework to analyze Integration and Test (I&T) data from three separate NASA spacecraft.**

***Keywords*- state-space coverage; verification and validation (V&V); t-wise testing; partial t-wise testing; configuration model; component interaction failure**

I. INTRODUCTION

The Verification and Validation (V&V) process for complex configurable systems typically involves system-level end-to-end scenario testing to demonstrate that the system satisfies requirements and meets customer needs. Since resources are limited in the V&V test environment, test procedures are generally restricted to demonstrating that the system meets the requirements in a limited number of configurations. The process of selecting just a few of the many possible system configurations to be tested is most often based on qualitative best engineering judgment. As a result, judging the quality of one V&V test plan in comparison to another is a difficult task. An obvious way to improve the quality system-level V&V testing is to increase the number of tested configurations, thereby providing an improved quantitative measure of how well the V&V test plan spans the system state-space.

The configuration state-space (referred to henceforth simply as the state-space) of a complex system is defined as the collection of all valid system configurations. Formally, for a system with n separate configurable components, the state-space is the set of all n-dimension vectors where each of the vector dimensions represents one system component and may take any valid numerically-represented value that expresses a configuration for that system component. The state-space coverage of a system level Verification and Validation (V&V) test plan is the subset of vectors in the state-space that are exercised in full end-to-end scenario-based V&V tests.

The motivation for studying state-space coverage of a test plan is two-fold. First, state-space coverage analysis provides a measure of the likelihood of uncovering unexpected system failures prior to deployment. Tests to demonstrate that the system satisfies requirements and meets customer needs often emphasize only common operational configurations and the extreme operating "corner cases." This leaves a large portion of the state-space unexplored, and subsequently any faults triggered by these unexercised configurations go undetected. This is particularly worrisome given empirical data supporting the hypothesis that system failures are generally triggered by a small number of components interacting in an unexpected way [1].

The desire to build systems that are operationally robust provides a second motive for analyzing state-space coverage. In many risk-adverse deployed systems, untested configurations are treated as operationally invalid, so the end-users are provided only a subset of potential system

978-1-4244-5883-7/10/$26.00 ©2010 IEEE

functionality. An understanding of the level and distribution of tested configurations translates into a measure of confidence in system performance for off-nominal states.

II. Modeling the System

In order to apply notions of state-space coverage to a complex hardware/software system, it is necessary to decompose the system into a finite vectorized configuration model such that any system configuration can be mapped to a configuration model vector. This is a multi-step process which requires selecting those model components that collectively represent the system under test, generating a discrete set of equivalence classes for any continuous-valued components in the system, and devising a method to convert whatever state-representation data is collected during V&V testing into configuration vectors in the state-space.

Decisions regarding the specific configurable components to be modeled depend on the overarching goals of the system-level test plan to be evaluated. Highly detailed models which define all of the controllable system elements are most robust, but generate increased state-space size which may translate into additional test cost. A much simpler model may be used when the testing goals are directed at a smaller subset of functionality. For example, many systems carry a number of redundant components which can be cross-strapped. A model consisting of only the system's redundant components may be used to explore the cross-strapping system state-space. These kinds of simplistic models may not cover every aspect of the controllable system state-space, but they are better suited to a system-level V&V test environment that seeks to demonstrate high-level interaction between major system components. System engineering judgment is required to determine an acceptable fidelity for the system state-space model.

III. State-space Coverage Analysis

Given a system that has been decomposed into an appropriate model, the most obvious quantification of model state-space coverage exercised by a system test plan is computed by dividing the number of state vectors explored by the system test plan by the cardinality of the entire state-space. However, for most complex systems, this method has limited usefulness since the combinatorial explosion of the state-space guarantees that the fraction of states covered during testing will be miniscule. This can be misleading given the interaction failure hypothesis described above. Additionally, this so-called *total coverage* metric contains no information about the distribution of states covered or the uniformity of testing across the entire state-space.

Since the main motivation to increase state-space coverage during system testing is to uncover more potential faults, one might consider studying how undetected faults are statistically correlated to various notions of state-space coverage. In fact, this kind of analysis has been an active area of research in the software testing community for some time. Kuhn, et al. determined that in nearly all cases software faults are the result of a small number (six or less in the cases studied) of configurable components or parameters interacting [2]. Proceeding from the hypothesis that modern software systems and software/hardware systems have a similar type of configuration and interaction complexity, in order to uncover all multiple-component interaction faults during testing, a test engineer would need to make sure that every collection of, say, six variables is tested in every possible 6-way configuration. This is certainly a less daunting proposal than testing every viable system configuration one-by-one, though it may still require a prohibitively large number of tests. In practice, however, a great majority of faults are triggered by smaller than 6-way interactions, so in many cases the number of tests required may be reduced.

This idea that failures are triggered by a specific configuration of a small number of interacting components leads to the notion of a *t-wise covering array*. For a system with n components, a t-wise covering array is a collection of test vectors such that for any selection of t components, each possible configuration of those components is realized by at least one test vector in the array. Contrast this with exhaustive testing, where each possible configuration of all n components must exist in the array of test vectors. Exhaustive testing (i.e. full state-space coverage) requires a number of tests that scales exponentially with number of components in the system model. A t-wise test suite, on the other hand, scales with the log of the number of components in current test-suite generation algorithms like the IPOG algorithm [3]. This is a great improvement in terms of cost and time for complex systems with many components. For example, in a system with n components, each of which can be in any one of k discrete states, the number of vectors in an exhaustive test array is

$$\mathrm{EA}(n, k) = k^n, \tag{1}$$

while the number of vectors in a t-wise covering array generated by the IPOG algorithm is

$$\mathrm{CA}(n, k, t) = O(k^t \log(n)). \tag{2}$$

Interpretation of a test suite within a t-wise covering array framework immediately implies a quantitative method for analyzing the uniformity of coverage of a test suite; computing the largest t for which the test suite forms a t-wise covering array provides a numeric result indicative of the likelihood that potential component interaction failures are discovered during execution of the tests. However, if the test suite being analyzed was not designed with explicit considerations towards t-wise state-space coverage, it is likely that the computed value from the t-wise analysis will be small. This means that the value assigned to most test suites falls in a very limited range, typically 0-2, so many test suites of varying efficacy are scored with the same value. Such a coarse scale does not lend itself well to quantitative analysis, particularly when performing comparative analysis between different test

suites. The next section describes new state-space coverage methods developed to overcome this limitation of *t*-wise coverage analysis.

IV. Partial t-Wise Coverage

In order to refine the values assigned to a test suite by *t*-wise analysis, we developed two notions of partial *t*-wise coverage. In the first, we sum the number of different *t*-length sub-vectors exercised by the test suite and compute the percentage of all possible *t*-length sub-vectors that are achieved. This value is termed the *total t-wise coverage* of the test suite. The second method involves observing each *t*-set of components one-by-one and determining what percentage of the total number of configurations for that *t*-set is achieved by the test suite. Choosing a value *q* in the interval [0, 1], the *(q,t)-completeness* value is the percentage of the *n-choose-t t*-sets that have at least a *q*-fraction of their configurations covered by the suite. Tables 1 and 2 provide a simple example. The *total 2-wise coverage* and the *(q,t)-completeness* for various values of *q* for this example model are compiled below.

- *total 2-wise coverage* = 19/24 = 0.79167
- *(.50, 2)-completeness* = 6/6 = 1.0
- *(.75, 2)-completeness* = 5/6 = 0.83333
- *(1.0, 2)-completeness* = 2/6 = 0.33333

TABLE I. Sample Test Array

a	b	c	d
0	0	0	0
0	1	1	0
1	0	0	1
0	1	1	1

The test array covers the six possible 2-wise combinations of a, b, c, and d to different levels, as shown in Table 2.

TABLE II. 2-wise Coverage for Sample Test Array

Var.	Configuration	Config. Coverage
ab	00, 01, 10	.75
ac	00, 01, 10	.75
ad	00, 01, 11	.75
bc	00, 11	.50
bd	00, 01, 10, 11	1.0
cd	00, 01, 10, 11	1.0

By fixing *t* and plotting *(q,t)-completeness* as a function of *q* for a given test suite, some useful data can be generated. *q*-values for which the *(q,t)-completeness* drops off steeply indicate important threshold values for the test suite. In particular, the largest *q* for which a test suite has *(q,t)-completeness* equal to 1 indicates the minimum percentage of coverage that any *t*-set in the test suite achieves. If this value is small, it means that the test suite is inadequately testing at least some of the *t*-sets. These *t*-sets can be identified at no added cost during the test analysis, and in cases where testing is still underway, the remaining tests can be designed to reduce or eliminate these deficiencies.

V. A Practical Example

To determine the practicality of analyzing existing test suites using partial *t*-wise coverage metrics, we examined test data collected during the Integration and Test (I&T) phase for three separate NASA spacecraft. The three missions were selected for analysis because of the availability of archived data and the presence of archived failure reports. This section outlines the procedure used to analyze I&T state-space coverage from a partial *t*-wise testing perspective. In each of the three cases, historical data was collected and analyzed long after the I&T phase; the I&T tests had been designed without any explicit consideration for *t*-wise coverage, so the exercise provided a realistic platform for analyzing and comparing state-space coverage of complete executed test sets.

The original tests had been designed using best engineering judgment to verify system requirements and validate day-to-day operations of the spacecraft. The test suites included "day-in-the-life" tests, fault scenario tests, and comprehensive performance tests designed to validate operations in both nominal and off-nominal states. They were not designed to achieve any specific quantitative state-space coverage values, though oftentimes considerations of whether a spacecraft was tested in various state configurations enters the discussion when during mission operations engineers must determine if a given achieved or desired configuration is safe for the deployed system.

The first step in analyzing the archived data was to build a system model for each of the three spacecraft. For practicality, we chose to create system models that contained less than 100 controllable system components. This provided sufficient complexity without making the process of gathering data and performing statistical analysis too burdensome. We selected those controllable elements for each of the three separate spacecraft that provided the most complete understanding of the system configuration within the desired model limits. For example, all hardware components that could be turned on and off were modeled as simple binary system components. High level software states that controlled the various major software algorithms were included as system components. Additionally, controllable interface parameters that determined which hardware-based sensor data is used by the software were included in the system model.

For each of the spacecraft, the I&T data had originally been archived as a telemetry time record series, a second-by-second snapshot of various subsets of the spacecraft system telemetry.

In order to convert telemetry time records into system model state vectors, a mapping was defined for each component in the system model; for every time record, each component in the model was assigned a state value based on the value of one or more telemetry points. In some cases the component state assignment was simply a one-to-one mapping from the value of a given telemetry point. Other cases were more complicated, requiring the evaluation of a series of Boolean expressions, each involving multiple telemetry points. As a simple illustration, consider a single system model component representing redundant hardware in the spacecraft. Redundant hardware components may report their respective states in different telemetry points, so in order to map a telemetry time record into a state for the model component, the telemetry point that indicates which of the hardware components is active is used as a switch to determine which telemetry point to use as the model component's state. See table 3 below for an example of the symbolic logical form.

The result of this mapping is a system model state vector for each of the telemetry time records. After each time record was processed, the resulting model state vector was compared to a lexicographically sorted array of previous state vectors, and inserted into the array if it was not already present. If it was present, a counter associated with the state vector was incremented. After processing all telemetry time records for a given spacecraft I&T data set, the result was an ordered array of system model state vectors and a single column vector representing the counts for each of the state vectors. This array was analyzed for total system state-space coverage, total *t*-wise coverage, and *(q, t)*-completeness for various values of *q* and *t*.

TABLE III. SAMPLE LOGICAL FORM FOR COMPOSITE CONFIGURATION VARIABLES

P: = Active Integrated Electronics Module (IEM) Component (0 or 1)
Q := Guidance Algorithm Model Component (1, 2, or 3)
S := Telemetry Point Indicating Value of Q in the active IEM (0 or 1)
val: S → Q
Compute:
(P == 0) → S = 1. ELSE S = 0. Q = val(S).

A. Data Caveats

It should be noted that not every telemetry time record was mapped to a system model state vector. In some instances the telemetry record was incomplete, missing one or more values for telemetry points required in component state calculations. This was expected given that various operational configurations for the spacecraft have restricted telemetry collection functionality and/or restricted downlink rates. A more unusual case of invalid telemetry involved one or more telemetry point values falling outside their legitimate range. This was attributed to either corruption in the retrieved data, failure in the data decomposition routines, or failure in the archival data collection routines. In all cases, each incomplete or invalid telemetry time record was discarded and no system state vector was generated for that time record. We did not record the percentage of telemetry time records discarded, though it was certainly minimal, probably less than 1% of the total time records evaluated.

For the spacecraft data evaluated, telemetry had originally been archived in binary downlink packet form. In order to generate telemetry time records, the data was decomposed using conversion files for the most recent flight software telemetry packetization routines. However, these files were not necessarily identical to the files valid at the time that the telemetry was produced, so there is some risk that portions of the telemetry were decomposed incorrectly. Given the difficulty in determining which conversion files were valid for various time segments during I&T, the difficulty in decomposing such a large set of data against multiple files, and the minor differences between those files used at the start of I&T and the current files, the authors decided that the risk of data corruption did not warrant the effort that would be required to eliminate this risk.

To reiterate, the primary purpose of evaluating archival spacecraft I&T data for state-space coverage was as a proof-of-concept, not as a hard analysis of the spacecraft I&T test plans. The considerations detailed above should alert the reader to the fact that there is some uncertainty in the completeness and validity of the quantitative results. Still, we present the results as an example of the kinds of analysis that can be completed using a *t*-wise testing framework for evaluation of state-space coverage.

B. Results

Prior to partial *t*-wise coverage analysis, some basic statistics about the data sets were computed. These results are tabulated in Table 4. Figures 1 – 3 show how the count values were distributed. Most of the state vectors were observed in less than 10 of the time records processed, though a number of vectors were observed in more than 10,000 time records.

As described above, it is difficult to make qualitative assertions about the state-space coverage given the miniscule numbers involved. For example, although the state-space coverage of Spacecraft 3 is three orders of magnitude smaller than those of Spacecraft 1 and 2, the values are so small that this difference conveys little useful information. The relative distribution of coverage is not apparent. *t*-wise coverage and partial *t*-wise coverage analysis provide a more extensive and communicable collection of metrics. These results are shown in the figures below.

TABLE IV. STATE-SPACE COVERAGE CALCULATIONS

Spacecraft	Num. of Model Components	Total State-space Size	Time Records Converted	Num. of Distinct State Vectors	State-space Coverage
Spacecraft 1	82	1.959×10^{26}	10323523	7617	3.889×10^{-23}
Spacecraft 2	82	1.959×10^{26}	9586265	7489	3.824×10^{-23}
Spacecraft 3	94	2.139×10^{30}	9574082	22541	1.0537×10^{-26}

A number of conclusions can be drawn from the results of the partial *t*-wise analysis of the three data sets. Interestingly, none of the data sets achieve even *2*-wise (pair-wise) coverage. This means that in all cases there is some pair of components that were not tested in all possible 2-way configurations. For a test engineer, identification of those pairs that are not covered could suggest deficiencies in the initial test set, and in cases where analysis is performed prior to the end of the system test phase, it might still be possible to test these missed pair-wise configurations.

Note that in the (*q,t*)-completeness plots, the *q*-values at which the curve drops steeply represent threshold values; there are a significant number of *t*-sets that are not covered beyond each of these threshold *q*-values.

Another interesting artifact of the analysis is that for all observed values of *t*, Spacecraft 3 remains near the optimal (*q, t*)-completeness value of 1 over a greater number of *q*-values than Spacecraft 1 or 2. This is particularly surprising given that the total state-space coverage for Spacecraft 3 is three orders of magnitude worse than for the other spacecraft. Evidently, testing of Spacecraft 3 was better distributed across its state-space than testing of Spacecraft 1 or 2, at least in the *t*-wise sense of state-space distribution.

Of course, it is important to keep in mind the concerns raised in the Data Caveats section, and to recognize that in this proof-of-concept, the systems were modeled more for clarity and ease of data retrieval than for implications of any state-space coverage analysis. There is still work to be done to correlate system tests that perform well in partial *t*-wise analysis with systems that are more robust and less likely to fail after deployment. Our limited data set analysis is not meant to provide that argument, but rather to prove that this kind of partial *t*-wise analysis is a tangible way to characterize state-space coverage of system tests and to perform comparative analysis for multiple test suites, even using old or archived test data.

VI. CONCLUSION

The *t*-wise framework provides a meaningful mathematical abstraction for performing state-space coverage analysis of existing V&V test sets. The data produced in partial *t*-wise analysis convey information about the amount of coverage and equally importantly the distribution of test vectors within a large state-space. This distribution is a key component of off-nominal validation, giving systems engineers, system test developers, and operators a way to quantify the breadth of system-level tests. This idea coupled with evidence from the software testing community that multiple-component interaction failures are a prevalent source of system errors suggests that *t*-wise analysis of test plans is a valuable means of test suite comparison.

By scoping the system model appropriately, analysis can be performed on state-space coverage at various layers of the system. Regardless of scope, the state-space, which grows exponentially as components are added, is analyzed for convergence to an ideal that only grows as the log of the number of components, so modest modifications to the test plan can be shown to make relatively large contributions to test effectiveness, and improving model fidelity by adding components is not as potentially detrimental to state-space coverage quantifications.

Analyzing older test data within a partial t-wise context can be a non-trivial exercise. Major difficulties may arise in converting the stored data to a series of system model state vectors. The important considerations here include model definition and completeness of archival data. If the model is deficient for the type of analysis being performed, the results may not accurately reflect the likelihood that component interaction failures were uncovered during testing. If the archived data is incomplete or difficult to retrieve, the analysis results may short-change the level of coverage actually achieved by the system tests. These issues are present in any sort of after-the-fact state-space coverage analysis, however, and are not restricted to partial *t*-wise analysis.

As systems grow ever more complex and configurable, methods are required to determine adequacy of system-level tests. In many instances requirements verification and validation of expected operating scenarios does not provide sufficient coverage of potential system configurations. Fault-tolerant systems, in particular, are designed to continue operating in off-nominal or unexpected situations, so greater emphasis must be placed on the distribution of configurations exercised during testing. A common framework for analysis allows apples-to-apples comparisons for choosing among competing test plans, and provides a means to compare different systems for adequacy in testing.

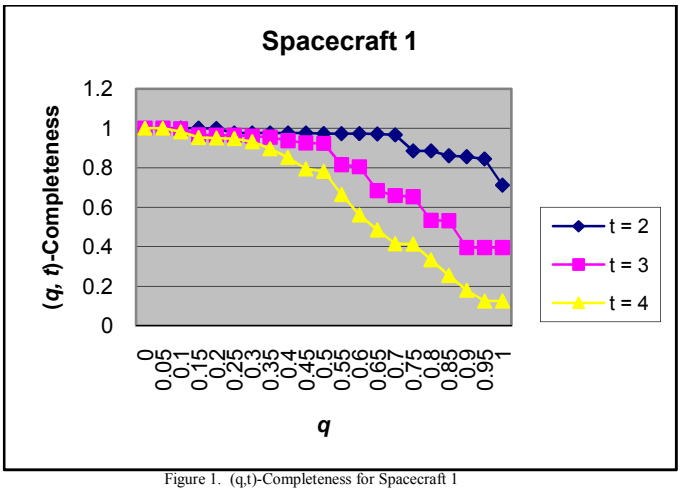

Figure 1. (q,t)-Completeness for Spacecraft 1

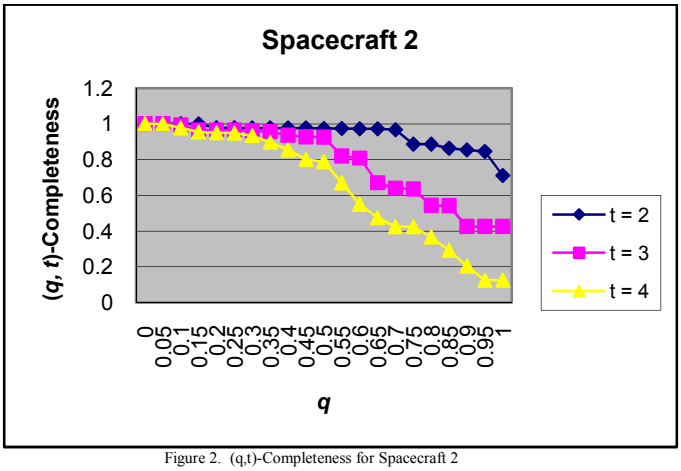

Figure 2. (q,t)-Completeness for Spacecraft 2

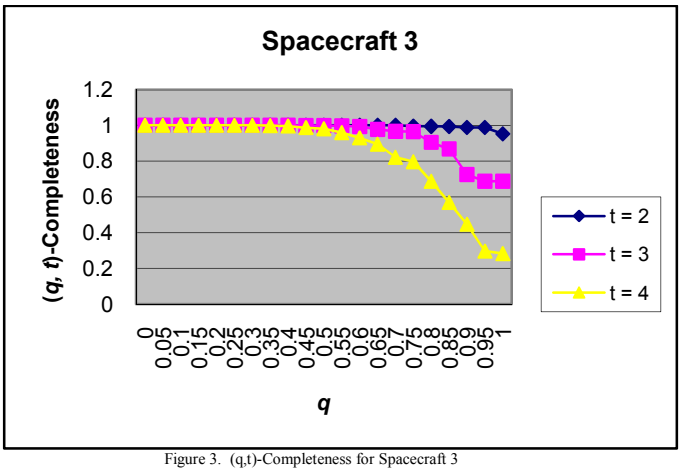

Figure 3. (q,t)-Completeness for Spacecraft 3

[1] C. Perrow, Normal Accidents: Living with High Risk Technologies, Basic Books, New York, 1984.

[2] R. D. Kuhn, D. R. Wallace, A. M. Gallo Jr., "Software Fault Interactions and Implications for Software Testing," IEEE Transactions on Software Engineering, Vol. 30, No. 6, June 2004.

[3] Y. Lei, R. Kacker, D.R. Kuhn, V. Okun, J. Lawrence, "IPOG – a General Strategy for t-way Testing", IEEE Engineering of Computer Based Systems conference, 2007.

[4] A. P. Godbole, D. E. Skipper, R. A. Sunly, "t-Covering Arrays: Upper Bounds and Poisson Approximations", Combinatorics, Probability, and Computing 5 (1996), 105-118.

[5] D.M. Cohen, S.R. Dalal, M.L. Fredman, G.C. Patton "The AETG System: An approach to Testing Based on Combinatorial Design." *IEEE Transactions on Software Engineering*, Vol. 23, No. 7, July 2007.