Practical Approach to Increasing State Space Coverage for System Testing

Michael D. Trela
Johns Hopkins University
Applied Physics Laboratory
michael.trela@jhuapl.edu
11100 Johns Hopkins Road
Laurel, MD 20723

Joshua Maximoff
Johns Hopkins University
Applied Physics Laboratory
joshua.maximoff@jhuapl.edu
11100 Johns Hopkins Road
Laurel, MD 20723

This paper presents a practical solution to improving system level Verification and Validation (V&V) tests, those tests which exercise end-to-end functionality of a physical system, for complex hardware/software systems. The proposed method to improve system level testing is to increase the total number of system configurations that are tested in a way such that the state space coverage is explored in a systematic and evenly-distributed fashion. We describe a method for adapting the combinatorial software test strategy known as t-wise testing to complex hardware/software systems V&V testing. Practical requirements and potential test selection architecture are provided to demonstrate the utility of the proposed t-wise testing method. Some methods for test selection and prioritization are outlined for instances when complete t-wise testing is impractical.

Introduction

System level Verification and Validation (V&V) testing serves a variety of purposes depending upon the program and system under test. Typically, system V&V testing provides an opportunity to demonstrate that the system satisfies its requirements (Verification) and the customer's needs (Validation) by executing operational scenarios. System-level scenario test plans are developed in order to show that the system meets a particular requirement or are based on the systems engineer's judgment of likely situations in which the system will operate. In the end, decisions regarding the selection of system level V&V tests are based on a quantitative assessment, which makes judging the quality of test plans difficult.

We proceed from the assumption that one important measure of the quality of a system level V&V test plan is its ability to test the system in various configurations across numerous scenarios. The state space coverage of a system test plan is described as the subset of all possible initial system internal and external configuration settings that are achieved by executing tests within the plan. To ensure the robustness of the system, it would be ideal to be able to test all possible system configurations in any number of operational scenarios. Unfortunately, the resources for many system level V&V test programs prohibit achieving this kind of full state space exploration.

For example, consider the development and testing of a new satellite. A typical satellite carries numerous levels of redundancy to reduce mission risk in the event of a single component failure. Assuming that the spacecraft has 10 redundant components, all of which are fully cross-strapped

(i.e. any redundant component can be used in place of the primary unit), it would require 1024 (2^10) tests to fully verify that the system can operate in any cross-strap configuration. Given the resource demands of a single test, such a large number of tests would be impractical for many systems. As a result, systems engineers typically execute a handful of system-level scenario tests that exercise a small number of configurations. The remaining system configurations are then "verified and validated" by drawing analytical conclusions from smaller (cheaper) compatibility and interface tests. Time and budget constraints ensure that not all spacecraft system configurations can be tested before launch.

We first present practical requirements for methods that increase the state space coverage during system level V&V tests. We provide an abstract description of t-wise testing and some notes on adaptability requirements. Then, by using the spacecraft Integration and Test (I&T) environment as an example, we show that t-wise testing has real world potential to improve system level V&V tests. We provide information and guidance regarding the various practical choices that must be made in order to model a system for a t-wise testing approach. Additionally, we describe a series of metrics that can be used to assess the quality of a V&V test plan in terms of its t-wise completeness. Discussions regarding the practical benefits over current state of the art approaches to testing as well as potential weakness are provided. The paper concludes with an outline of potential refinements to t-wise testing using a graph-theoretic modelling approach.

Practical Requirements for Increasing State Space Coverage during V&V testing

Given the numerous stakeholders involved, limited programmatic resources, and relatively high-risk nature of system level V&V testing, several practical requirements must be considered when proposing a new method to increase the state space coverage of a test suite.

Communicability of the test plan is paramount. System level V&V is a high profile event during the system development life-cycle at which time a large number of stakeholders (test conductors, management, sponsors, sub-system technical engineers) with numerous, varied agendas will influence the test design. To ensure a proposed test strategy meets the needs of all stakeholders, any practical approach to increasing the state-space coverage for V&V tests must be easily communicated to those stakeholders. The decision process to select some test configurations over others needs to be clear and transparent. If some particular complex combinatorial method used to select the various configurations under test is difficult to explain to the diverse stakeholders, the unfamiliarity of the proposed method will render its adaptation for actual use unlikely. Confidence in a new method or procedure requires a clear, intuitive explanation before the testing community will accept.

Additionally, any new method to increase the state space coverage of a test suite must not result in significantly increased test time. System level V&V testing often spans only a small set of the complete system state space due to limited programmatic resources (cost and schedule). Any new method for increasing the state-space coverage of a system level V&V test plan requires a careful optimization of the testing resources for it to be accepted by the systems engineering community at large. Adding additional tests which only demonstrate system robustness within a larger tested state space does not have sufficient and direct measurable benefit. Absent a measure of earned-value, additional testing costs are difficulty to justify. For example, Monte Carlo methods used to seed test configurations for computer models are difficult to adapt for physical

hardware/software system testing given the large number of tests required to yield useful statistical averages. Therefore, the number of tests (or test resources) that are to be used by a new testing methodology should not increase and ideally would decrease over time.

Finally, system-level V&V testing is usually a highly risky venture that requires testing methods that are unambiguous to the test conductor and easily repeatable. Whether flight testing a new airborne weapons system or conducting mission simulations for a deep space satellite loaded with propellant, there is an absolute need to know the complete system configuration to safely execute end-to-end V&V testing. Moreover, if an anomaly or failure occurs during a system level V&V test, it must be possible to execute the exact same test to demonstrate repeatability of the error and to prove out any fixes that may have been implemented.

Introduction to T-wise Testing

Given a system model consisting of n independently configurable components, a t-wise test suite (equivalently t-wise covering array) is a collection of test vectors such that for every subset of t components there are vectors in the set which exercise every possible configuration of those components. In other words, every sub-vector of length t is expressed in some test vector in the array. Execution of a t-wise test suite guarantees that for any choice of t components, every t-way configuration of those components is tested. Compare this with an exhaustive test suite, which contains a vector for each of the possible configurations of all n components. The exhaustive test suite requires one test vector for every state vector in the system state space, a number that grows exponentially with the number of configurable components, n. A t-wise test suite, on the other hand, requires far fewer test vectors when t is much smaller than n. In fact, fast algorithms exist [6] to produce t-wise covering arrays with a number of tests on the order of k^t log_2(n), where k is the number of possible configurations for each component. The logarithmic factor of n is a great savings over the exponential factor for exhaustive test suites in systems with many configurable components.

There is a growing body of empirical evidence that most system failures are the result of complex interactions between subsystem components [8]. The number of components involved in these types of failures is an active area of research. Kuhn, et al. studied a collection of complex, configurable software systems [4] and found that no more than six components were involved in any reported failure, with most failure attributable to four or less components. Given similarities between complex software systems and software/hardware systems, it is reasonable to assume that these numbers will prove valid in the hardware/software domain. We are particularly interested in collecting empirical data for a variety of hardware/software systems as this work goes forward.

Under the assumption that system failures are often attributable to a small number of components interacting in an unexpected way, t-wise testing provides a means to uncover many system failures during the test phase. Since a t-wise test suite guarantees that all t-way configurations are fully tested, any failure triggered by a configuration of t or less components will be exhibited during execution of the test suite. T-wise testing provides a systematic way of exploring the limited interaction sub-state space of the system, providing a balance between cost (as quantified by the number of system level tests) and likelihood of finding potential failures during testing.

Practical Use of T-wise Testing

While the use of t-wise testing has been successfully demonstrated in the Verification and Validation of software systems [1], it can also be extended to complex systems such as a spacecraft undergoing a system level Integration & Test (I&T) program. To demonstrate the potential benefit and real applicability of t-wise testing to a hardware/software software system, we conducted a study to examine the state space coverage of some past spacecraft I&T tests in terms of t-wise coverage [7]. To adapt the spacecraft data for analysis we established a framework to discretize the state vectors of a hardware/software system and developed a series of metrics by which one can evaluate how close a test plan comes to achieving t-wise completeness. In the following section we present a potential application of t-wise testing to the spacecraft I&T environment that increases the testing state-space coverage without running additional tests.

Spacecraft I&T refers to one of the final stages in a space vehicle development program where the various system components and subsystems are fully integrated with each other prior to launch. During spacecraft I&T, system level tests are executed to ensure subsystem compatibility, evaluate total system performance, and perform mission simulations that simulate nominal and off-nominal scenarios. The end result of spacecraft I&T is to verify that it will meet its system level requirements and validate its performance will meet the sponsors' needs. Given the large number of staff (i.e. high cost) that are needed during spacecraft I&T, testing time is extremely limited. Any proposal to increase the state space coverage of system level V&V tests by simply adding additional tests without a measurable earned-value would not be accepted in the space community. Therefore, employing a t-wise test strategy that seeks to minimize the required test covering array for spacecraft I&T could be a solution to increasing state space coverage without violating program resources.

Any given system-level spacecraft test will typically only test a small handful of variables while leaving the rest of the system in a default configuration. For example, a thermal engineer may run numerous tests on the spacecraft verifying heaters, temperature sensors, and thermostats are working properly at the integrated system level. During this period, other non-interfering subsystem components such as Command & Data handling software (C&DH) or communication radios would likely be relegated to a single default state as the operating status of these components does not impact the primary test article. A more optimal solution, one which better utilizes the limited spacecraft test time, would systematically place the non-interfering components into different configurations to increase the system level state-space coverage.

The use of a t-wise test strategy to implement a systematic approach to selecting the states of non-interfering controllable components (i.e. variables) is ideal since it provides clear state space coverage milestones. Figure 1 below depicts the proposed spacecraft I&T approach to increasing state space coverage via t-wise testing. First, a test conductor provides a list of components that have a required fixed state for the test. The required system test configuration is then passed to an online tool aptly named a "t-wise test generator." The online tool determines the states for each of the "free variables" based on the required component configurations, previously executed tests, and any other goals for prioritizing the orders of tests (to be discussed further). Then, the test controller approves or rejects the proposed full system test configuration. Finally, the operator configures the spacecraft components in their appropriate states and executes the desired test. During the course of the test, spacecraft telemetry is fed back into the online database updating the list of previously tested configurations. From a conceptual level, the only additional requirements

to the spacecraft I&T program is the development of the “t-wise test generator and database” and the review of proposed system configurations by the test controller.

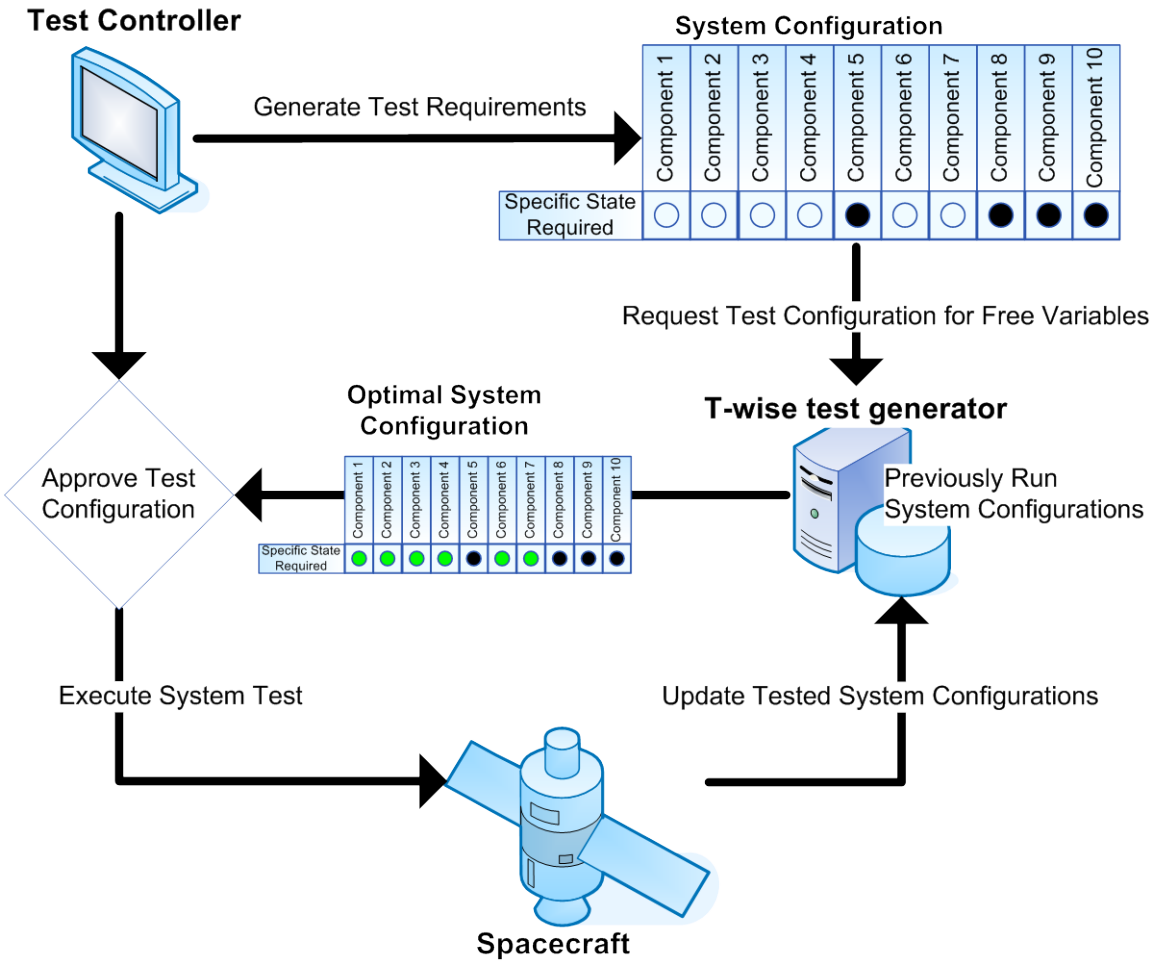

Figure 1. Schematic of Proposed T-wise Testing Strategy

Modelling the State Space

In order to adapt t-wise testing to a practical hardware/software system, it is necessary to consider the various options for modelling the system state space. In the strictest sense, the purpose of testing across a system’s state space is to see how the system will respond when its controllable elements are placed in various operating states. Therefore, when modelling a state space for t-wise testing, only those system components which can be directly controlled should be included in the model. The selection of controllable elements to include the in the system’s state space model must be made based upon the desired testing goals. For spacecraft and other highly-redundant systems there are a number of ways to build up a state space model.

The most basic and highest level option would be to simply model the redundant components. A typical deep space or large GEO stationary spacecraft carries redundant components (e.g. primary and backup heaters) to minimize the risk of a single point failure ending the mission. Given the typical number of redundant components, properly testing all the possible system cross-strapping

configurations is a very costly and impractical task. The number of configurations tested generally constitutes a very small fraction of the set of all configurations. Since many missions choose to restrict the operating configurations for the deployed system to only those that were tested during I&T, systems are often *operationally* constrained by I&T limitations. T-wise testing provides the system engineer a way to systematically increase the number of tested configurations, thereby reducing future operational constraints. Furthermore, if the engineering team accepts the notion that failures are resultant of no more than t components interacting, then if the redundant components are t-wise tested, the system can operate in any desired configuration.

A more refined level of modelling would include high-level state variables of both non-redundant and redundant components. The high-level state variables should provide a general picture of the current system status and potential emerging behaviour. The purpose of this modelling is to ensure at a high level the various system components do not have any major gross interactions and the system is able to successfully execute basic functions. For a complex system like a spacecraft this level of modelling would produce approximately 100 system state variables.

An even more detailed approach to robust testing of the system state space would be to discretize *all* of the system control variables and execute a full t-wise test array. Applications of t-wise testing in software verification and validation typically use this approach. All of the software parameters are discretized and a full t-wise covering array is used to set the initial conditions and configurations of the software as regression tests are performed. In a spacecraft, or similar complex hardware/software system, this may not be a feasible option give the many hundreds or thousands of control commands that may exist. Instead the system engineer can take a hybrid approach by modelling elements whose control commands are expected to be used on a regular basis. At this level, the goal of state space exploration is to verify and validate that the system is robust enough to operate in all of its various nominal operational configurations.

Ordering of tests to achieve levels of T-wise completeness

Achieving full t-wise completeness for systems of moderate complexity (~100 components) and t values up to 6 is practical for faster than real time software and simulations, but still is unlikely to be achieved by a hardware/software system like a spacecraft. Take for a example a system with 50 components, each of which has 3 states. The smallest t-wise test suite generated by current state-of-the-art algorithms has more than 8000 test vectors; this is far outside the scope of a typical spacecraft I&T test campaign. Therefore, metrics that determine the level of t-wise completeness are useful to provide the status of the V&V state space coverage. The metrics can also be used by a system as presented in Figure 1 to guide the selection of new tests.

As presented in a previous conference proceeding [7], the authors developed a set of metrics to represent t-wise completeness. For a given value of t and a test suite that does not represent a complete t-wise covering array, the notion of partial t-wise testing is useful. Indeed, system testers may want to know the likelihood that faults triggered by more than t interacting components will be uncovered by executing the test suite. Two metrics associated with a partial t-wise covering array are particularly useful: The first, referred to as the total t-wise coverage, is the percentage of t-length sub-vectors that are contained within the array. To compute this value, simply count how many different t-length sub-vectors are in the array, where two sub-vectors are different if either they are indexed by a different set of t columns or they are indexed by the same t columns but differ in at least one component. The second metric is parametrized by a value q from the interval [0,1] and is referred to as the (q,t)-completeness of the test suite. It represents the percentage of

t-sets for which at least a q-fraction of the possible configurations exists in the array. To compute this value, enumerate the n-choose-t t-sets, and then go one-by-one through these sets determining if at least a q-fraction of the possible configurations are expressed. Divide the number of t-sets that are at least a q-fraction covered by the total number of t-sets, n-choose-t. The total t-wise coverage and (q,t)-completeness computations can be repeated for any number of values of t and q, and the results can be used to form a quantitative view of the likelihood that interaction failures involving various numbers of components will be uncovered by executing the test suite.

It is expected that initial adaptations of t-wise testing will work to achieve a high level of partial t-wise coverage without increasing the number of tests in the I&T test plan.. Take for example the proposed spacecraft I&T t-wise test generator. This online utility receives from the test conductor a set of test requirements, references the previously run test configurations and makes a decision on how to configure the remaining free variables. To determine which configuration adds the most value, the t-wise test generator might select the configuration from all remaining vectors in a t-wise covering array that is most orthogonal to the previously run tests (i.e. greatest component-by-component dissimilarity). This type of scheme would then provide the greatest span of the V&V testing state space but may not achieve complete t-wise coverage if time constraints force fewer than the requisite number of tests. Another scheme could focus on making certain that particular sets of critical system components achieve high (q,t)-completeness as compared to less critical components. An even simpler process for determing the next test vector would be to consider only a smaller set of the variables states to ensure a minimum (q,t)-completeness. These examples serve to illustrate that the choice of optimization routines to prioritize t-wise covering arrays under the assumption that full t-wise completeness will not be achieved is extensive.

Benefits of T-wise testing

In many instances, using t-wise testing to increase the state space coverage of V&V system testing is a practical systems engineering approach having clear benefits over the current practice of best system engineering judgment. In order to evaluate whether a given system will benefit from this kind of testing paradigm, one must consider the operational environment of the system and whether it is required or likely that system will operate in numerous configurations. In a highly risk-adverse environment like human space flight, it may be required that every system configuration be tested exactly how it will be flown, limiting the number of configurations that are considered valid post-deployment. However if the hardware/software system will be part of a larger network (e.g. a system of systems) which has an extraordinarily large operating state space, systems engineers need to solve the “2^N” problem to ensure the system will perform as intended. Currently, systems engineers systematically plan V&V system tests based on the criteria such as most probable or most severe expected operating conditions. System configurations that are not exercised in an end-to-end system level test, are then verified and validated for use by analysis rather than system testing – which does not reduce mission risk as much as an actual test. T-wise testing provides a way for systems engineers to move beyond best engineering judgment.

It should be noted that t-wise testing is one of a number of combinatorial method that seeks to provide intelligent coverage of the system. These methods, categorized as deterministic, non-deterministic, or compound (deterministic and non-deterministic), have been shown to achieve similar results [3]. However, the deterministic nature of t-wise testing allows users to generate a complete test suite and then prioritize the order of execution according the specific

system-testing needs (sub-configurations) or goals (work towards the highest level of t-completeness) in those instances where complete t-wise coverage cannot be achieved. As compared to other combinatorial approaches, t-wise testing is a highly intuitive construct that can be easily communicated across a multi-disciplined team. Describing the motivation behind a t-wise strategy is simple: test all the possible configurations between every small set of t variables.

The most practical argument for using t-wise testing may be the extensive research performed by the software testing community considering its efficacy and developing automated test-generation tools. As mentioned previously, NIST has investigated numerous software applications and determined that the total number of variables that lead to failures is less than six. Although hardware/software systems have additional complexity including non-discrete operating conditions, testing of these systems is nominally performed in a discrete fashion. System level verification tests will typically test extreme operating points (ex. Hot and Cold temperatures) and verify the system can operate in any point in between by analysis. Additionally, the research, development, and accessibility of efficient algorithms to generate compact t-wise covering arrays can be leveraged to quickly deploy t-wise test strategies to hardware/software systems. While numerous combinatorial methods to explore a system's state space exist, many of these are either infeasible (i.e. algorithms are too complex to implement in actual software) or still in the very early stages of development. The ACTS utility, among others, demonstrates an efficient way to produce compact test arrays for software V&V and can be easily adapted by the systems engineering community.

Weaknesses of T-wise testing

Despite its potential to be readily adapted by the systems engineering community for system level V&V testing, t-wise testing does have several drawbacks that should be examined in future research. First and foremost, adaptation of t-wise testing to hardware/software systems requires the decomposition of the system into components that assume discrete configuration states. Since t-wise testing is rooted in testing all the possible distinct combinations of states between t variables, a system that cannot be decomposed into discrete states would not be suitable for this form of test. Another area of concern is the potential for a particular test vector to mask the behaviour or responses of other variables. In the generation of a t-wise covering array, algorithms build up test vectors based on the need to satisfy t-wise combinations of variables and do not consider the entirety of the system configuration. Therefore, if the system behaviour is defined by more variables than the chosen t value there is potential of not testing the state spaces thoroughly enough.

Another aspect of adapting t-wise testing is addressing the question, "what constitutes a successful test of a t-wise test vector." In hardware/software systems, like the spacecraft I&T environment, it is impractical to execute a full set of performance regression tests for every tested configuration. This so-called "oracle problem" is recognized by practitioners of t-wise testing in the software community, though many proposed solutions [5] are impractical in the hardware/software domain due to real-time execution considerations. The "crash-testing" approach, for example, simply considers whether the system seems to have survived a tested configuration. However, in a real-time system a crash may not be immediate. One option may be to consider the time associated with potential failures for a given component state. Take for example a system component A that is in state 0 and the system engineer judges that any anomalies from this operating condition will arise in less than n seconds, and classifies this as the (A,O) failure time. An online utility that

records test system configurations could record how long each component state was maintained and if no anomalies arise before the maximum of all (A,O) failure times over all component configurations O in the test, then the configuration was successfully tested. However, this may re-introduce best engineering judgment in the failure time determinations, trading one type of best guess for another. In any case, there isn't a clear answer to the question and further research into the successful test of a t-wise test vector should be conducted.

Potential refinements to t-wise testing

In many instances, certain sets of system components are known to interact weakly or not at all. Traditional t-wise testing would still test for interaction failures involving these components. As discussed, system testing resources are generally scarce, and in situations where component interaction likelihood can be characterized, it may be useful to spend resources on those tests where the probability of triggering interaction failures is greatest. One approach is to adopt a graph model of the system and perform tests only to cover those t-sets of components that exhibit a certain graph property.

Recall that a graph G is a pair (V,E) where V is a set of vertices and E is a collection of edges between pairs of vertices. A system can be modelled as a graph where the vertex set V is simply the same collection of configurable components that we have been discussing and the edge set E is given by drawing an edge between two components whenever there is a direct interface between them. For example, the main processor in a system and the data disk are connected by an edge because they share an electrical (and a data) interface. With the system graph model defined in this way, we may define a (G,t,r)-wise test suite as a collection of test vectors such that every t-set of components with the property that the set has pairwise distance at most r in the graph G has every possible t-way configuration covered by at least one vector.

It is not difficult to show that any algorithm which can produce a t-wise covering array of size f(t,n) can easily be modified to produce a (G,t,r)-wise covering array of size at most f(t, $\chi(G)$r), where $\chi(G)$ is the chromatic number of the graph (i.e. the smallest partition of the vertices of G into sets such that no two vertices in the same set are connected by an edge in G. It is well known that $\chi(G) <= \Delta(G)$, where $\Delta(G)$ is the maximum number of edges incident to a single vertex in the graph [2]. Recalling that best known algorithms for t-wise covering arrays produce test sets that scale as log_2(n), for systems without no component having more than (log_2(n))r interfaces, this can amount to a savings. The trade-off, of course, is that only t-sets of components that are pairwise interface-reachable in r or less steps are guaranteed to be tested. This may at least serve as another method for addressing the test prioritization problem outlined in the "Benefits of t-wise Testing" section.

Conclusions

Using the t-wise test concept developed by the software community, this paper proposes a method to increase state space coverage during system level V&V testing. The overarching principle of the proposed approach is to develop a system to manipulate "free variables" during required or pre-planned system V&V tests to uncover potential failures and demonstrate system performance across a larger subset of the state space. The proposed test architecture for the spacecraft I&T environment provides a simple example of how t-wise testing could be adapted to an actual system test plan. Further information is provided on how tests could be prioritized, and how notions of

partial t-wise coverage could be applied in situations where a full t-wise test suite cannot be executed. Additional research and development are required before deploying the concept to an actual hardware/software system, but the foundational algorithms and test-generation software packages have already been developed by the software testing community.

Ultimately, improving system level V&V testing is a difficult task since most measures of improvement are largely subjective in nature. The authors suggest that if free-variables can be exploited systematically on a test-by-test basis to exercise an increased number of configurations, then the opportunity to uncover potential failures during system V&V testing is increased. Although, changing the processes and procedures used by system-level V&V test engineers is a difficult task, a t-wise test strategy is a manageable step forward, since it can be easily communicated to various stakeholders, does not significantly increase required test time, and poses no great risk to the system under test. Compared to industry standard "best engineering judgment" test strategies, t-wise testing provides a quantifiable, more systematic approach to validating complex systems.

References

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

[2] Diestel, R., 2005. Graph Theory, Third Edition. Heidleberg: Springer-Verlag.

[3] Grindal, M, J. Offutt, S.F. Andler. 2004. Combinatorial Testing Strategies: A Survey. In GMU Technical Report ISE-TR-04-05, July 2004.

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

[5] Kuhn, R.D, Y. Lei, R. Kacker. 2008. Practical Combinatorial Testing: Beyond Pairwise. In IEEE IT Professional, vol. 10, no. 3, June 2008.

[6] Lei, Y, R. Kacker, R.D. Kuhn, V. Okun, J. Lawrence. 2007. IPOG - a General Strategy for t-way Testing. In the Proceedings of the 14th Annual IEEE Engineering of Computer Based Systems Conference (Tuscon, AZ). IEEE.

[7] Maximoff, J, M. Trela, R.D. Kuhn, R. Kacker. 2010. A Method for Analyzing System State-space Coverage within a t-Wise Testing Framework. To be included in The Proceedings of the 4th Annual IEEE International Systems Conference (San Diego, CA). IEEE.

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

Biography

Michael Trela is a systems engineer in the space systems applications group of the Johns Hopkins University Applied Physics Laboratory. He worked as satellite systems engineer for NASA's STEREO and MESSENGER missions. His experience includes satellite concept development, fault management engineering, spacecraft autonomy systems, and propulsion analysis. He graduated from Stanford University with a MS in Aeronautics and Astronautics and the University of Notre Dame with a BS in Aerospace Engineering.

Joshua Maximoff is a spacecraft software engineer in the embedded applications group of the Johns Hopkins University Applied Physics Laboratory. He develops hardware-in-the-loop simulators for test and evaluation of NASA spacecraft, including MESSENGER, STEREO, and Radiation Belt Storm Probes. He received a MA in Mathematics from Arizona State University and a BS in Mathematics from the University of Nevada.