Validation and Verification of LADEE Models and Software

Karen Gundy-Burlet[1]
Intelligent Systems Division, NASA-Ames Research Center, Moffett Field, CA, 94035

The Lunar Atmosphere Dust Environment Explorer (LADEE) mission will orbit the moon in order to measure the density, composition and time variability of the lunar dust environment. The ground-side and onboard flight software for the mission is being developed using a "Model-Based Software" methodology. In this technique, models of the spacecraft and flight software are developed in a graphical dynamics modeling package. Flight Software requirements are prototyped and refined using the simulated models. After the model is shown to work as desired in this simulation framework, C-code software is automatically generated from the models. The generated software is then tested in real time Processor-in-the-Loop and Hardware-in-the-Loop test beds. "Travelling Road Show" test beds were used for early integration tests with payloads and other subsystems. Traditional techniques for verifying computational sciences models are used to characterize the spacecraft simulation. A lightweight set of formal methods analysis, static analysis, formal inspection and code coverage analyses are utilized to further reduce defects in the onboard flight software artifacts. These techniques are applied early and often in the development process, iteratively increasing the capabilities of the software and the fidelity of the vehicle models and test beds.

I. Introduction

The Lunar Atmosphere Dust Environment Explorer (LADEE)[i] is a small explorer class mission that is currently scheduled to launch in late summer 2013. It will orbit the moon to determine the density, composition and variability of the lunar exosphere. To minimize fabrication and design costs, the "modular common bus" spacecraft was designed with common structural components that could be connected together to form the spacecraft. LADEE is formed of four such modules, housing the propulsion, avionics and three instruments for observation of the moon. The mission will also perform a technology demonstrate of a laser communications device. For further details on the mission and spacecraft, please see Hine, Spremo, Turner and Caffrey[ii].

With LADEE physical construction proceeding down a low-cost rapidly prototyped product-line type of effort, a similar philosophy drove the development of the software base. Traditionally, flight software represents one of the schedule and cost drivers for spacecraft systems. A recent GAO[iii] assessment of large-scale space science projects noted that the Mars Science Laboratory slipped development and test of the entry, descent, landing and surface

[1] LADEE Flight Software Lead, M/S 269-3, NASA-Ames Research Center, Moffett Field, CA, 94035. Associate Fellow, AIAA.

operations software into the operations phase while the spacecraft was transiting to Mars. Because of the, the operations phase of the mission encountered significant cost increase, of which "additional $59 million to ensure achievement of mission success criteria and accommodate development of surface mobility flight software". While the strategy was in the end successful for MSL, many other missions have been canceled or had catastrophic failures associated with the flight software. A survey by Hayhurst and Holloway[iv] identified several driving factors leading to large-scale software development problems such as these, including lack of knowledge of best practices in software engineering, communications difficulties between regulators and among project personnel and the difficulty of clearly specifying requirements. Other extensive analyses with sets of recommendations exist in the literature[vvi] and will not be repeated here. With strict launch deadlines and a short 100-day mission, LADEE could not afford to slip software delivery in any way and developed a strategy to tightly integrate V&V in the early development cycle to minimize risk.

Successful examples of rapidly deployed small-spacecraft missions were surveyed. Of these, the AFRL XSS-10 and XSS-11 spacecraft utilized model-based development methodologies, and we performed a trade study on the methodologies using a hover test vehicle[vii]. One advantage with model-based development is that controls engineers are able to rapidly prototype algorithms in their "native language" and software engineers could directly autocode the models. This minimizes the opportunity for communication errors between algorithm designers and qualified software developers. The model based-methodology also enhances early prototyping of requirements, enables validation and verification during early stages of development and provides a common platform for communication between subsystems, software engineers and stakeholders. This paper discusses the range of V&V techniques used in order to provide a reasonable expectation of success for a mission with tight schedule and budget requirements.

II. Software Overview

LADEE utilizes a model-based development approach layered on hand-code, Government-off-the-shelf (GOTS) and Commercial-off-the-shelf (COTS) software elements. The core of the spacecraft control and simulation software is modeled in Mathwork's Simulink[viii] software. Figure 1 shows that LADEE models are broken into two major components – Onboard Flight Software (OFSW) and the Ground Side Simulation Equipment (GSSE). The OFSW is modeled independently of the simulation of the spacecraft and are autocoded separately for use on different processors in our hardware test environment. The simulation of the spacecraft was developed to sufficient fidelity to develop and test the flight software (it is not intended to be an authoritative source of the behavior of the vehicle). It includes environmental effects,

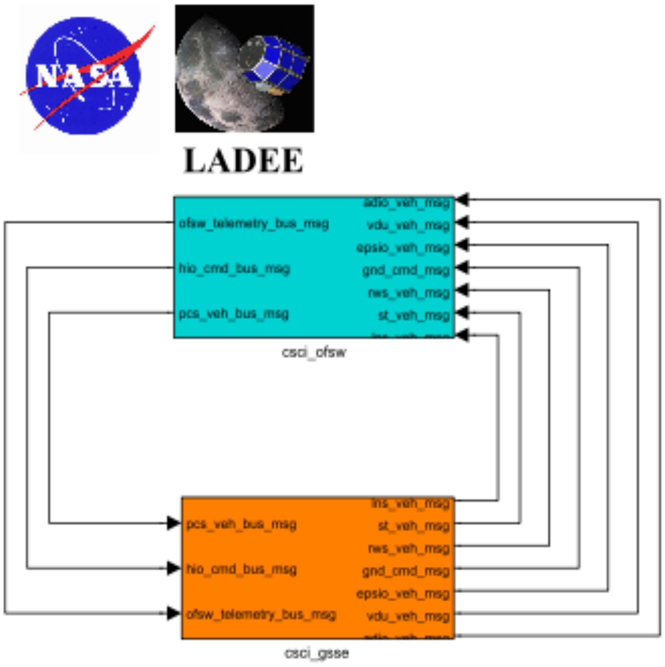

Figure 1. Top level model diagram showing Onboard Flight Software component (teal) and simulation of spacecraft (orange).

spacecraft dynamics, models of thermal, electrical, power and propulsion systems as well as the ability to insert failures in the spacecraft systems.

The overall layered architecture for the onboard flight software is shown in Figure 2. The applications autocoded out from Simulink modules are shown in yellow, with each control application being coded as a separate object. These are integrated with the Core Flight Executive and Core Flight Software[ix] (blue) using a hand-coded "Simulink Interface Layer". Other hand-code developed for the project includes memory scrub, telemetry, command and hardware input/out drivers shown in green. The real time operating system used here is VxWorks. One requirement on the architecture of the flight software was that each component be amenable to V&V techniques such as those described here.

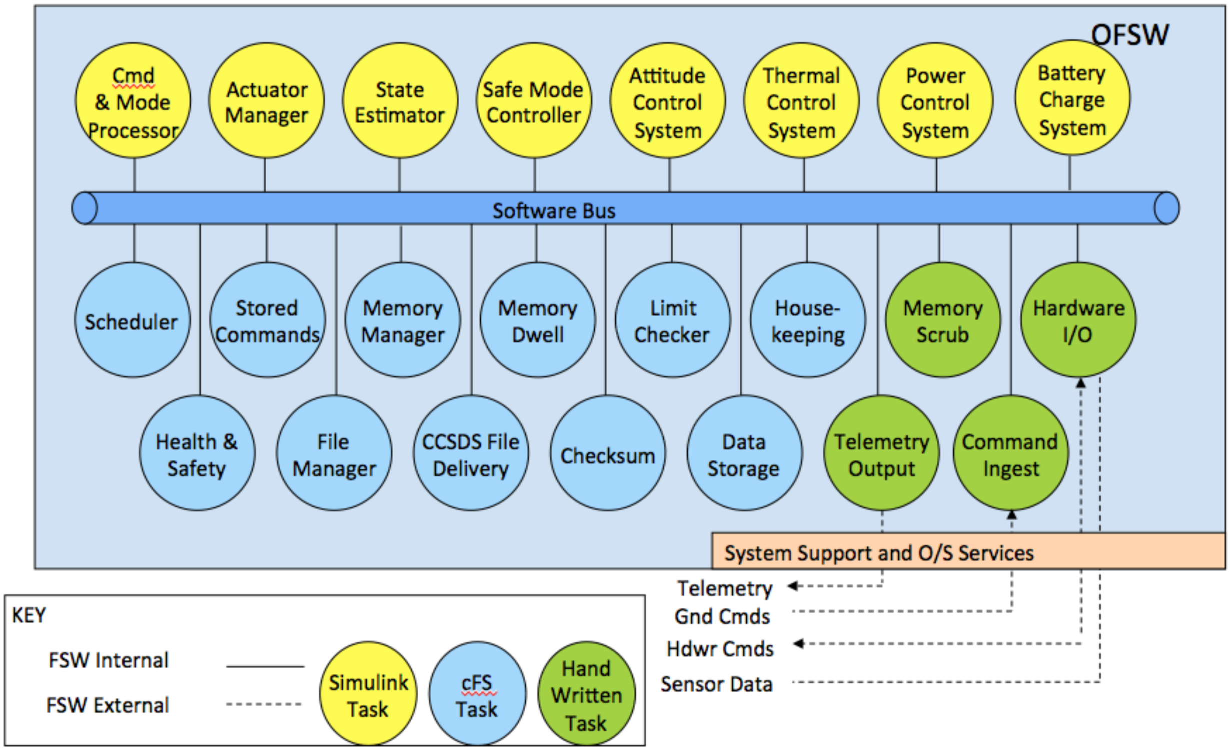

Figure 2. Flight Software Architecture

III. Definitions

There is substantial controversy over the exact definitions of the words Validation and Verification across the range of scientific computing literature. Oberkampf and Trucano[x] has an extended discussion of the way the terms are used by AIAA and the Computational Fluid Dynamics (CFD) community while the National Academy of Sciences[xi] extends the terms to be applicable over a wide range of scientific computing. In general, these communities define verification as how well the computational model reflects the governing equations that were solved while validation determines how well the computational model reflects reality in the range of parameters of interest. These definitions are sensible in a community where formal software engineering practices such as maintaining strict requirements on behavior of software are not typically utilized. Instead, the governing equations and specification of the algorithm become, in effect the source of requirements. For development of flight software, there are formal NASA Software Procedural Requirements[2] (NPR 7150.2) that define validation and verification activities as:

- “Software verification is a software engineering activity that demonstrates the software products meet specified requirements”
- Software validation is a software engineering activity that demonstrates the as-built software product or software product component satisfies its intended use in its intended environment

With this definition, the V&V activities on the simulation software resemble those of the greater computational sciences community. The V&V activities on the OFSW follow the NPRs for onboard flight software development.

The NPR is an extensive document that outlines the general software engineering practices that should be defined by the developing organization at each stage of the software life cycle. It does not specify particular techniques or practices, but instead guides the software development team to consider practices to avoid failures that have been encountered in other onboard flight software development efforts.

IV. Requirements

Core to the software development effort are specifications of clear and concise requirements on the behaviors to be verified. Too often projects over specify requirements, allow turbulence in the formulation of requirements or accept “desirements” as real requirements. As the LADEE software development process was stood up early in the project lifecycle (before the physical system specifications) were known, we used the opportunity to prototype candidate requirements along with the system models and software.

A key concept was that the requirements identified the critical behaviors that needed to be tested at each level of the flight software. One tendency projects have is that elements of design are often documented in a design document and then also instantiated in the requirements. This leads to an extensive set of requirements in which the key drivers of system performance are hidden among design elements. Design documents are equivalent in importance with formal requirements and the as-built code/model is inspected relative to the as-designed to rectify any differences. On LADEE, a conscious decision was made to formulate requirements to be performance-related test drivers to scope the testing activities.

On the LADEE mission, the Level-4 requirements governed the behavior of the integrated flight software (e.g. “The OFSW shall…”), while Level-5 specified the behavior of each subsystem (e.g. “The Attitude Control System shall…”). These could be further decomposed to specify unit level behavior. The testing strategy for each of these levels of requirements is described below.

V. Simulation hardware

Several different test environments were utilized depending on the level of requirements being verified and focus of the test being applied: Workstation Simulation (WSIM), Processor in the Loop (PIL) and Hardware in the Loop (HIL) systems were used in different phases of development and testing. The WSIM environment includes only the Simulink modules. It is used early in development process to perform algorithm development, requirements analysis and unit testing model performance. For the PIL environment inexpensive “flight-like” processors

[2] NPR 7150.2, http://nodis3.gsfc.nasa.gov/main_lib.html

were utilized. The Simulink models were autocoded and integrated with running with the C&DH software running on the real-time processors. PILs were useful for testing the software integration, timing and communications and the impact of communication delay on control algorithm performance. The HIL systems incorporated flight avionics Engineering Development Units (EDUs) that provided definitive answers on integrated performance, resource utilization and were the highest fidelity simulations for V&V activities.

One important risk reduction activity was the development of the "Travelling Road Show" (TRS) where an EDU would be integrated with the flight software in a mobile chassis. Members of the FSW team would travel to sites where payloads and other subsystems were being developed to test the interfaces long before integration of the hardware on the spacecraft. These tests clarified the Interface Control Documents and payload requirements and caught defects at a time when they could be more easily fixed. One lesson learned from LADEE is that where we did not perform TRS testing, we had critical defects found in spacecraft integration that had to be rapidly corrected and verified.

VI. Unit Testing and Rollup of Information

The development effort on LADEE emphasized testing of the software at the proper level. It is usually hard to test and debug internal modes and interfaces at a system level and impossible to test system integration issues at a unit level. It can be expensive and difficult to debug latent defects in models that arise during integrated testing. Thus, an early emphasis was placed on the unit test suite infrastructure with an eye toward permanently capturing the unit test effort as a regression test suite.

Testing on the LADEE program is enhanced by the modular and layered system architecture. The NPRs specify that one must maintain bidirectional traceability between code/models, requirements, design and test artifacts. This was performed in a low overhead manner using a system of naming conventions. For example, for the modeling environment, pertinent naming conventions are:

- <name>_lib.mdl: Simulink Model library
- <name>_hrn.mdl: Simulink test harness
- <name>_test.m: Test script driver associated with model library.

Where the <name> included a unique identifier for that model library that could be cross-references with requirements and other external documents. Each developer was responsible for providing all the necessary artifacts and developing the unit test suite associated with their model libraries. By adhering to the naming conventions, higher-level regression test suites could interrogate the LADEE model and generate a manifest of all expected test artifacts. This manifest was used to exercise the entire test suite under development and evaluate the progress on verifying requirements. Simulink Report Generator was used as a platform to both drive the test suites and capture the resulting information in a manner that provided bi-directional traceability between low-level requirements, models, test suites and metrics.

This test suite also exercises the GSSE side of the LADEE model. We've chosen our requirements to document the comparison of the simulation results with that of analysis or similar simulation or test results from the spacecraft itself. Assumptions are documented as well as ranges of acceptable operation for the model. Depending on the type of model, refinement studies, sensitivity analyses, comparison with analysis, subsystem performance specifications or outside recognized tools are included.

To reduce the software development schedule and budget, LADEE incorporates a significant amount of GOTS software from the cFE and cFS packages developed at Goddard Space Flight Center. There is an extensive set of documented test results for each unit of the software that was considered sufficient verification evidence. It was not considered a value-added exercise to independently re-execute the provided test suite on each isolated component. Instead, we are concerned with the operation of each re-used unit within the context of the complete integrated flight software.

To explore the interactions among modules, we have a constantly expanding test suite that checks the behavior of individual commands in the Command and Telemetry Dictionary (C&T) in the full up flight environment including interactions with the ground station software (ITOS). While this is conducted with the full integrated flight software, the test suite is most often used for verification of Level-5 requirements, so we identify it more closely as a unit test suite. We've identified patterns of testing and implemented an automation language for writing Spacecraft Test and Operation (STOL) scripts. In order to provide traceability, the data from the test suite is associated 3 ways: relative to requirements, on a per command basis and on a per test basis. Metrics are provided on each basis to assess the progress of the command coverage and verification of Level-5 requirements. As results from the test suite are interrogated to provide evidence for higher-level requirements, they are output in parse-able html files.

VII. Preparation for Software Integration Testing

The unit test suites are used as early indicators that pertinent requirements are being met, but they do not test the integrated functionality of the software. That is done in a PIL/HIL environment through the use of scenario-based testing. A spacecraft concept of operations was developed that looked at each phase of the life cycle and scenarios were developed to test anticipated operations. These included separation and activation, science operations, orbital maneuvering and fault management related scenarios. The concept of operations specified which requirements would be met in which build cycle. Basic GN&C and software functionality was allocated to build 1 while full fault-management functionality was allocated to build 4. Build 5 was reserved for reduction in software defects and expanding payload interfaces. In each build cycle, additional test scenario scripts were written for each of the new requirements that were to be addressed. A test harness is then used to put the spacecraft in the appropriate mode for testing, issue the appropriate command scripts for the flight software and simulation sides, telemeter the data off of the spacecraft at the end of the scenario and post process the binary data files into comma-separated value files as well as textual log files. The specific files needed for verification of the level-4 requirements are copied into a database for further post processing.

This is a highly I/O bound process, and several strategies were used to optimize the wall-clock time associated with it. Initially, complete bus messages were downloaded for post-processing, but much of the data was only used internally on the spacecraft, not for the level-4 verification. Hand-encoded "housekeeping" packets proved useful for minimizing the amount of data brought off the spacecraft, but the process of coordinating the telemetry record files (.reek) with display pages (.page), sequential print formatting (.sport) and the "C" code that copied specific telemetry locations into the new packets was time consuming and highly error prone. A Matlab script was developed that auto generated the rec, page, sprt and copy_table.c files given a simple text specification of the telemetry point and original record it could be found in. Despite the growth in the number of scenarios over the build cycles, use of the housekeeping telemetry packets enabled us to plateau at only about 70Gb of data for verification of the level 4 requirements. While useful for minimizing the data foot print for FSW testing, the routines have the added side-benefit of making it easy for the mission operations team to optimize packets that fit within the narrow bandwidth requirements for safe mode telemetry.

Once the scenarios have been run, a custom Simulink report generator script is used to post process the data and capture the output. The script first reads external imported data, such as requirements spreadsheets and Interface Control Documents. It then processes each of the csv files to extract needed variables for the test suite. This is once again an I/O intensive process, so if a fresh set of csv files is found, a Matlab binary file is written with the extracted information. The data is further compressed by only storing change-points in the data. For busses with mode information or slowly changing status bits, significant compression can be achieved. On successive runs of the test suite, the extracted binaries are used in the initialization process, but the original csv files are retained in order to facilitate the debugging process if errors are found.

Each chapter of the report incorporates one level-4 requirement and runs the associated level-4 test. Naming conventions are once again used to associate the various artifacts. Each of the test scripts has a common interface to assist in the automation of the system. In particular, a structure containing the entire processed scenario data is passed into the test routine, and a text message, test status and the handle to an html file are passed back out of the routine. Any images that were generated during the course of the test are captured and incorporated in the output of the chapter. A summary verification matrix is also generated that is extremely useful for quickly identifying failures within the extensive documentation generated.

VIII. Test Patterns for verification of Level-4 requirements

The flight software Level 4 requirements are directed at the operation of the system and the requirements statements are crafted to be verifiable using formal methods. Optimally, one would utilize formal methods on the underlying Simulink model to ensure that the logic would always provide the expected results. However, the Simulink model is only one layer of the flight software, and it is the propagation of the signals through the interface layer and the core services that are of interest for verification of the overall flight software. Current state of the art in formal methods is typically directed at early models of system behavior (e.g., the Simulink models), and formal analysis of fully integrated flight software remains intractable. The telemetered data represents the authoritative source for verifying that the OFSW performs as required in the context of the mission. Thus, concepts from the formal methods communities were utilized, but applied to the telemetered data stream in a lightweight manner suitable for regular regression testing.

Typically, the requirements were encoded as Matlab strings in a top-level script. The encoded assertions would be evaluated on the telemetered data using dynamic programming techniques. Status of the test would be returned along with evidence supporting the summary status. One subtlety associated with correlating telemetry is the clock stamp behavior on each of the telemetry packets. The telemetry is time stamped as it is generated within each application and different applications run at different times. There is dither and drift in the clocks between the flight processor and the simulation. Occasionally telemetry can be lost, and as stated previously, the telemetry is compressed to change-points. The data reduction algorithms have been designed with these conditions in mind, but occasionally false positives occur, usually due to a telemetry point being on the cusp of the decision logic as to whether it is a member of the telemetry set or not. These are generally resolved by inspection of the underlying data. In many early cases though, failures of the assertion indicated mode logic problems within the flight software itself.

Several different classes of testing were abstracted from the form of the requirements, and fall into the major categories below:

1. Guidance, Navigation & Control (GN&C) related limits

GN&C requirements typically state that in a particular mode that the absolute values of an error term not exceed a limit. The scripts gather the maximum value of the error term for all pertinent scenarios and test platforms and the limit criterion is applied.

2. Commanding, Modes and Transitions

Requirements in this category typically specify behaviors such as "while in a flight mode, with active control disabled, a particular parameter shall also be disabled" or "on transition to a particular mode, a parameter shall be disabled". A simple Matlab-based temporal language was developed to specify the required behaviors and underlying scripts applied the assertions to the telemetry stream.

3. Fault Management

The spacecraft simulation model was developed with hooks to test various fault management scenarios that were considered to be correctable by flight software. A series of scenarios was been set up to with commands sent to the simulated spacecraft that ranged from out of range temperatures to stuck thrusters. A fault management spreadsheet was developed which identified critical signals, latency, persistence and the type of response. A recursive algorithm was used to find the period of time in which all signals were true, and scanned the log files to find the expected response. The time stamps were compared to determine if the response happened within an acceptable window. One complication is that if the software transitioned to safe mode, telemetry rates are restricted and data from the critical signals could be lost. In this case, the software monitors to see if the desired response was found in the telemetry even if the trigger could not be specifically identified. These were identified as passing tests, but with a note that they were demonstration only.

4. Commands and Telemetry

Several of the level 4 requirements are directed at ensuring that variables are in metric, commands have explicit states, or that certain items of telemetry exist. Generally, these sorts of "inspection" requirements would be delegated to a review process for verification, but given the vast amount of information to be reviewed, and effort was made to automate and regression test these requirements. Regular expressions were extensively used to correlate variables, commands and telemetry with interface control documents that were used to identify required behavior. This turned out to be a remarkably effective technique to monitor design and requirements changes in the lower level software.

IX. Formal Inspection

Formal Inspection has been a useful technique to drive out defects and promote consistency among the various software artifacts. Rather than just walking through code, reviewers are assigned to various roles, and asked specific questions to drive their reviews of the material. Advanced software engineering tools were utilized to focus the inspection. Results of a static analyzer would be used by a reviewer to direct attention to unproven aspects of the code. Code coverage analysis would aid in identifying missing test cases or requirements. Code and models would be compared to style guides and the underlying design documentation to ensure consistency. After the designer removed defects found in the inspection the moderator would point-by-point review the changes with the designer and track compliance. Through the flight software development process, we formally logged a total of approximately 1150 hours of inspection time with a cumulative defect rate capture of about .65 per hour.

X. Conclusion

Budget, schedule and the complex nature of flight software make complete testing of the system an intractable problem, but the criticality of the operations demands a well-rounded approach to validation and verification. The process to properly verify flight code has foundations in techniques utilized by the computational sciences community as a whole. Additional rigor is directed by the nature of the different requirements on the operation of flight software vs. traditional models and simulations. Static analysis, code coverage, simplified formal method techniques, extensive test automation with automatic documentation and formal inspection were utilized to create a tractable validation and verification program within the context of a flight mission.

Acknowledgments

K. Gundy-Burlet thanks the LADEE Flight software team for their outstanding effort in support of this project.

References

[i] Delory, G.T.; Elphic, R.; Morgan, T.; Colaprete, T., Horanyi, M;, Mahaffy, P.; Hine, B.; Boroson, D, “The Lunar Atmosphere and Dust Environment Explorer (LADEE)”, 40th Lunar and Planetary Science Conference, (Lunar and Planetary Science XL), held March 23-27, 2009 in The Woodlands, Texas, id.2025

[ii] Hine, B., Spremo, S., Turner, M, Caffrey, R. “The Lunar Atmosphere and Dust Environment Explorer (LADEE) Mission”, NASA Technical Report/Patent Number: ARC-E-DAA-TN1098.

[iii] “NASA Assessments of Selected Large-Scale Projects”, GAO-12-207SP, March 2012.

[iv] Hayhurst, Kelly J., and C. Michael Holloway. "Challenges in software aspects of aerospace systems." Software Engineering Workshop, 2001. Proceedings. 26th Annual NASA Goddard. IEEE, 2001.

[v] Dvorak, D. (editor), “*NASA Study on Flight Software Complexity”*. NASA Office of Chief Engineer, *2009.*

[vi] Duren, R. M. “Validation and Verification of Deep-Space Missions”, JOURNAL OF SPACECRAFT AND ROCKETS Vol. 41, No. 4, July–August 2004.

[vii] Hine, B., Turner, M., Marshall, W. S. , “Prototype Common Bus Spacecraft: Hover Test Implementation and Results”, NASA/TM-2009-214597.

[viii] Simulink, Matlab, and M. A. Natick. "The Mathworks." Inc., Natick, MA (1993).

vii Wilmot, J. “Implications of Responsive Space on the Flight Software Architecture”, 4th Responsive Space Conference, RS4-2006-6003, April 2006.

[x] Oberkampf, W.L. and , Trucano, T.G., “Verification and Validation in Computational Fluid Dynamics”, *Progress in Aerospace Sciences* 38 (2002) 209–272

[xi] National Research Council. 2012. Assessing the Reliability of Complex Models: Mathematical and Statistical Foundations of Verification, Validation, and Uncertainty Quantification. Washington, D.C.: The National Academies Press.