

V&V of ISHM Software for Space Exploration

Lawrence Markosian, QSS Group, Inc./NASA Ames Research Center
Martin S. Feather, Jet Propulsion Laboratory, California Institute of Technology
David Brinza, Jet Propulsion Laboratory, California Institute of Technology
Fernando Figueroa, NASA Stennis Space Center

NASA has established a far-reaching and long-term program for robotic and manned exploration of the solar system, beginning with missions to the moon and Mars. Integrated System Health Management (ISHM) will be key to improving the reliability, operability and maintainability of many of the systems deployed in this endeavor.

The requirements of the ISHM systems themselves will be subject to requisite levels of Verification and Validation (V&V) and certification. The factors that most influence ISHM's V&V and certification needs stem from two main sources – the system of which ISHM is a part, and the implementation of ISHM itself. The system of which ISHM is a part levies requirements on ISHM – for example, the need for ISHM to respond within a given time period with a stipulated level of confidence in the correctness of its response. These *externally* imposed requirements have far reaching implications for V&V and certification of ISHM. The combination of these requirements, coupled with the manner in which ISHM will be utilized, drive much of the V&V and certification process. Also highly influential is the nature of the ISHM implementation. Often it takes a combination of techniques to implement an ISHM system. These techniques include well-understood algorithms for low-level data analysis, validation and reporting; traditional capabilities for fault detection, isolation and recovery; and, at the more novel end, AI techniques for state estimation and planning. Detailed descriptions of these techniques are beyond the scope of this paper, and may be found elsewhere. Here we focus on their ramifications for V&V and certification. In particular, this range of techniques that will be utilized within an overall ISHM system impose *internal* challenges to V&V.

The conjunction of these externally and internally influences on ISHM V&V and certification, and the challenges that stem from them, is the focus of this paper. We outline existing V&V approaches and analogs in other software application areas, and possible new approaches to the V&V challenges for space exploration ISHM.

1 EXISTING SOFTWARE V&V AND CERTIFICATION PRACTICES

There are many areas other than NASA where embedded systems exhibit a safety critical role, for example Avionics, Defense, Medical Devices, Nuclear Power, and Transportation. In this section we begin by looking at existing V&V and certification practices as seen in a representative one of these areas, Avionics. This area has many parallels to the safety- and mission-critical needs that predominate in NASA's space activities. We then suggest that the existing NASA hierarchy of requirements, policies, standards and procedures relevant to software have close parallels with those seen in the other safety critical areas.

1.1 Avionics V&V and Certification

Safety-critical software for commercial aircraft undergoes certification by the FAA, which includes V&V in accordance with RTCA/DO-178B. This document is recognized as the means for evaluating software for compliance with the relevant Federal Aviation Regulations/Joint Aviation Regulations (FARs/JARs) for embedded systems in commercial aircraft. A useful paper providing interpretation of RTCA/DO-178B was prepared by a Boeing participant in the RTCA committee responsible for DO-187B. The paper describes the intent and rationale of DO-178B. The derivation of the software approval guidelines from the Federal Aviation Regulations (FARs) to DO-178B is discussed in the paper to clarify its relationship to the government regulations. An explanation of the Designated Engineering Representative (DER) system is also provided in the paper along with a discussion of the safety process to describe the environment in which DO-178B is used.

The DO-178B/ED-12B Software Verification Process defines specific verification objectives that must be satisfied; these include:

- a. Verification of software development processes,
- b. Review of software development life cycle data,
- c. Functional Verification of software
 - i. Requirements-based testing and analysis
 - ii. Robustness testing
- d. Structural Coverage Analysis

Verification of the software development processes is accomplished by a combination of reviews and analyses. For software requirements, these include reviews of the quality of the requirements themselves, a requirements trace from system-level to low-level (code), and checks of their compatibility with the hardware; verifiability; conformance with standards; accuracy, correctness and behavior of algorithms. The software architecture is reviewed and analyzed for compatibility with the high-level requirements and target hardware. Conformance of the software architecture to standards, verifiability, consistency and partitioning integrity is also reviewed. The source code is also subjected to compliance and traceability to requirements. Conformance of the source code to standards, code verifiability, accuracy and consistency are also reviewed and analyzed. The integration process is verified by examination of the data and memory maps (detect memory overlaps or missing components).

DO-178B section 11 stipulates a number of data requirements: plans, standards, procedures, and products (including the source code and executable code) that document this certification. These are:

- Plan for Software Aspects of Certification
- Software Development Plan
- Software Verification Plan
- Software Configuration Management Plan
- Software Quality Assurance Plan
- Software Quality Assurance Plan
- Software Requirements Standards
- Software Design Standards
- Software Code Standards
- Software Requirements Data

- Software Design Description
- Source Code
- Executable Object Code
- Software Verification Cases and Procedures
- Software Verification Results
- Software Life Cycle Environment Configuration Index
- Software Configuration Index
- Problem Reports
- Software Configuration Management Records
- Software Quality Assurance Records
- Software Accomplishment Summary

The review of software development life cycle data includes assessment of the test results, configuration management and quality assurance aspects for the development. The testing portion, due to its complexity is described in detail below. The control of the configuration of the software, including identification of configuration items, establishment of configuration item baselines, change control data, traceability throughout the development cycle is reviewed and analyzed. Problem reporting, tracking and corrective action records are reviewed for adequacy and verification of the change is confirmed via examination of configuration records. The software quality assurance records are reviewed to provide confidence that the software life cycle processes have been followed and that deficiencies encountered in the life cycle are detected, evaluated, tracked and resolved.

Functional verification of the software is performed at three levels. (1) Hardware/software integration testing is performed to verify the correct operation of the software in the target computer environment. (2) Software integration testing verifies the interrelationships between software requirements and components and the implementation of the software components within the architecture. (3) The low-level testing verifies the implementation of software low-level requirements. These requirements-based tests are performed to verify correct functionality of the software in both normal range test cases and in robustness test cases. The normal case tests utilize valid and boundary values for inputs and exercises the transitions possible in normal operation. The robustness test cases inject invalid input values, values that would generate arithmetic overflows or attempt to provoke transitions that are not allowed. The software should follow expected behavior for the abnormal cases.

Structural coverage analysis is generally perceived to be the most difficult task to undertake in the testing process. Furthermore, certifying real-time executable code with an operating system that is tightly integrated with the hardware, cache, interrupts, memory management, and process/task management, can make structural testing even more difficult. These low-level aspects create a significant challenge to the verification process. Three primary levels of structural testing are invoked according to the criticality level of the software (Table 2) in DO-178B certifications:

- Statement Coverage (SC): Every statement in the program has been invoked or used at least once. This is the most common use of the term “code coverage.”
- Decision Coverage (DC): Every point of entry and exit in the program has been invoked at least once and that each decision in the program has been taken on all possible (Boolean) outcomes at least once. Essentially, this means that every Boolean statement has been evaluated both TRUE and FALSE.
- Modified Condition Decision Coverage(MCDC): Every point of entry and exit in the program has been invoked at least once, that every decision in the program has taken all possible outcomes at least once, and that each condition in a decision has been shown to independently affect that decision's outcome. Complex Booleans need to have truth tables developed to set each variable (inside a Boolean expression) to both TRUE and FALSE.

In DO-178B terms, software has a criticality level, ranging from the most critical (“Level A), down to “Level E”. Level A software requires *all three* levels of structural testing be performed.

Performing this code coverage exercise is possible using manual methods, but this process is now readily facilitated by utilizing commercial code coverage tools. Numerous code coverage tool vendors now supply testing tools that create the appropriate test outputs to demonstrate and satisfy compliance with DO-178B.

1.2 NASA Requirements, Policies, Standards and Procedures relevant to Software

The current NASA Software Safety Standard is NASA-STD-8719.13b, dated July 8, 2004, which applies to all safety-critical software acquired or produced by NASA. By reference this includes NASA Software Assurance Standard, NASA-STD-8739.8, dated July 28, 2004. This in turn includes by reference NASA NPR 7150.2, Sept. 27, 2004. The latter defines “Class A Human Rated Software Systems” as:

Applies to all space flight software subsystems (ground and flight) developed and/or operated by or for NASA to support human activity in space and that interact with NASA human space flight systems. Space flight system design and associated risks to humans are evaluated over the program's life cycle, including design, development, fabrication, processing, maintenance, launch, recovery, and final disposal. Examples of Class A software for human rated space flight include but are not limited to: guidance; navigation and control; life support systems; crew escape; automated rendezvous and docking; failure detection, isolation and recovery; and mission operations.

The classifications in NPR 7150.2 are important because, inter alia, the software engineering requirements, including V&V, depend on the classification. ISHM software

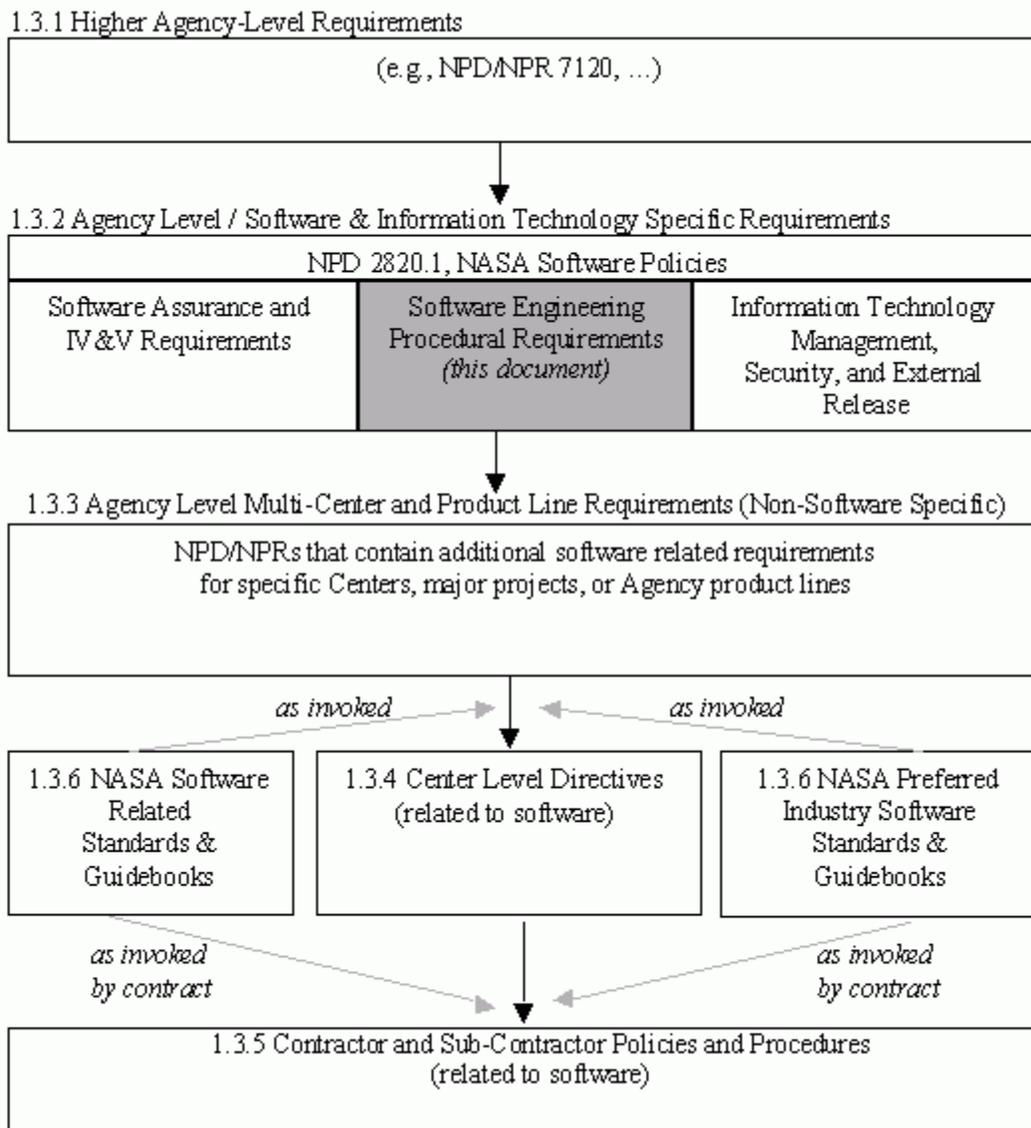


Figure 1 - Relationships Among Governing Software Documents (from NPR 7150.2)
is clearly Class A by this definition.

NASA's document NPR 7150.2, NASA Software Engineering Requirements, "provides a common set of generic requirements for software created and acquired by or for NASA..." Included in this document is a summary of the requirements with respect to software created and acquired by NASA. Figure 1, taken from this NPR, shows the relationships among the various relevant NASA requirements, policies, standards, procedures and guidance.

The net result of these governing documents is an approach to V&V and certification that has close parallels with those followed in other safety-critical application areas. Indeed, NASA's software working group is developing mappings between the NASA Software Engineering Requirements, NPR 7150.2, and select industry standards. A mapping to NASA's Software Assurance Standard exists, and (at the time of writing), mappings to the Software Engineering Institute's Capability Maturity Model Integration® (CMMI®), and to the Institute of Electrical and Electronics Engineers standard IEEE 12207, are "currently under review".

1.3 V&V for Spacecraft Fault Protection

Fault Protection software on existing NASA robotic spacecraft is a special case of ISHM. In general, ISHM goes beyond such Fault Protection in two major aspects: the need for reasoning, primarily as a consequence of the state-space explosion, and, in many applications, the focus on maintaining capability rather than the simpler task of averting catastrophe. Nevertheless, it is worth first considering how V&V is performed for Fault Protection before turning attention to ISHM in general.

Ideally, the development process of a spacecraft's Fault Protection would start with a detailed fault tree/FMECA effort that produces a clear "fault set". A fault set is the list of faults that the spacecraft or system might be subjected to that can then be subdivided into a "protected fault set" and an "unprotected fault set". In order to establish which is which, the project needs a clear definition of the project's fault tolerance is needed - is it to be single or dual fault tolerant? is the requirement to be fault tolerant or failure tolerant? etc. Having this fault set early in the life of the mission would provide the groundwork for the design and for risk trade offs as the hardware is selected. It would also provide a basis for the amount of redundancy selected for the hardware. Once the fault set is determined the fault injection requirements can be specified for the ground support equipment to be used to test the hardware and software.

This would be the ideal approach – however, in practice, this rarely occurs in its ideal form. As helpful as it would be to have the full fault set early in the mission, the project often does not have resources to dedicate systems engineers to a thorough fault tree and FMECA effort in early design. Usually, one gets *either* a fault tree *or* a FMECA *drafted*. This means that in practice there is a initial fault set but it is often very partial. The same is true of the fault injection requirements, which in practice, will in the initial stages be only a partial set.

To mitigate this reality, the best way is to ensure that both the fault set development and fault injection requirements identification are on-going processes with milestones at PDR, CDR and individual Fault Protection reviews so that the process can be kept somewhat current.

Finally, the FP testing process is itself constrained by project priorities. There is a theoretical desire to begin FP testing early and have it stay in step with the other s/w and

h/w testing. However, in practice the Fault Protection testing starts out with low priority, increasing as the overall testing program matures. Logic dictates that in a prioritized environment, there is no need for fault protection testing until the core nominal h/w and s/w is working. As the testing progresses and confidence in the nominal system matures, then attention turns to the off-nominal cases in which fault protection plays a central role.

Fault Protection testing has the same three levels of V&V as the other areas. It begins with – verifying the basic functionality of the fault protection software itself, that is, the fault protection governing software and the monitors and responses. One of the detailed methods used to accomplish this is to “enable” the monitors as soon as possible after a flight software delivery to ensure maximum testing time of the ability to detect errors. The remediation functions are exercised later in the test process as they become available. This testing can range from basic fault testing to a more extreme “stress testing” that involves cascading faults, envelope testing and heavy concurrent load testing. The stress testing completes the triage of verifying requirements, validating capabilities and then stress testing to find out where the system truly fails.

2 FEASIBILITY AND SUFFICIENCY OF EXISTING SOFTWARE V&V PRACTICES FOR ISHM

In this section we consider whether the existing software development practices can feasibly be applied as-is to ISHM systems, and whether those practices would provide sufficient levels of confidence in ISHM systems.

2.1 Feasibility

NASA’s Human-Rating Certification process is defined in NPR 8705.2A (effective date: 2/7/2005). The objective of the human-rating certification process is to document that the critical engineering requirements, health requirements, and safety requirements have been met for a space system that provides “maximum reasonable assurance” that the system's failure will not result in a crew or passenger fatality or permanent disability. This NPR covers numerous aspects of certification, including certification of software. One of the software aspects covered is testing, where one requirement is:

1.6.7.1 The Program Manager shall perform testing to verify and validate the performance, security, and reliability of all critical software across the entire performance envelope (or flight envelope) including mission functions, modes, and transitions.

ISHM clearly contains “critical software” and hence is subject to this testing requirement. However, the very nature of ISHM poses significant challenges to meeting this requirement, above and beyond challenges shared by most forms of mission-critical software. Specifically, ISHM, by definition, deals with *off-nominal* conditions in each of its roles (it must recognize, diagnose and respond to: early indications of impending failure, the presence of performance degradations, and failures that have occurred). Several V&V challenges stem from this: it is hard to know that all the significant possible failure modes have been identified (especially for relatively novel components and for conventional components operating in novel conditions); for any given failure mode, its

characteristics may not be well understood; there is a large number of ways in which off-nominal conditions can arise (consider all the parts that could fail, and the varying implications of such failure depending on when in the mission it occurs), and the combinations of such failures are vastly more numerous. For example, if there are 1,000 individual possible failures, then there are 1,000,000 *pairs* of such failures. This has specific relevance to the feasibility of meeting fault tolerance requirements that may be applicable. For example, another Human Rating requirement states:

Requirement 34419. Space systems shall be designed so that no two failures result in crew or passenger fatality or permanent disability.

In more general terms, the challenges posed by ISHM systems is that it is hard to assure completeness of models of failure, it is hard to assure that those models are correct, and it is hard to test/inspect/review the very many failure scenarios. While any given failure scenario may itself have a very low likelihood of occurrence, ISHM must be prepared to deal correctly with whichever ones do manifest themselves in the course of the mission, so V&V must address a large fraction of these to achieve the levels of assurance required.

In response to these questions of feasibility, the response could be to evolve the requirements, standards, etc. accordingly, or to leave them as-is and instead rely on provisions for exceptions¹, deviations² and waivers³ from these requirements. In practice waivers are common. Since they contradict the intent and effect of requirements, and introduce inconsistencies in the certification process, it is preferable to recognize early-on which requirements cannot be met, and revise these requirements as necessary to preclude reliance upon waivers.

2.2 Sufficiency

Another question to ask of the existing standards is whether they are sufficient to achieve the levels of assurance desired of ISHM systems.

We begin by noting that even the most stringent of the structural testing levels – the Modified Condition Decision Coverage (MCDC), cannot fully test a realistic software application. To do so would require “path” coverage, which is by no means guaranteed by MCDC. In MCDC each condition is tested largely independently of other decisions in the program, and in a program with n decision binary decision points there are 2^n independent decisions, each of which defines a possible path through the program. Of these, the number that are “feasible” (that is, that can actually be executed by some combination of input data values) is also on the order of 2^n . Thus only a relatively small portion of the possible execution paths are tested even under MCDC. For event-driven (reactive) systems the situation is even worse. ISHM systems fall squarely into this category. As described in the previous subsection, the number of possible behaviors can be a huge number, the small proportion covered by MCDC would leave an even larger number untested.

¹ An *exception* to a requirement can be provided if that requirement is not applicable to every component of the system.

² A *deviation* from a requirement can be provided if the requirement cannot be met but there is an alternative method of reducing system risk to an “equivalent or lower” level.

³ A waiver of a requirement may be requested if the requirement is unsatisfied and there is therefore an increased risk.

Further challenges stem from the unusual structure of ISHM software as compared to the more traditional forms of spacecraft software for which the standards, etc., were crafted. ISHM software often makes use of Artificial Intelligence techniques, and is architected accordingly. Specifically, such software typically has both a large, complex “reasoning engine”, and “*models*” (e.g., a model might describe the operating modes of the telecommunications system) over which that reasoning engine operates.

The implications for V&V are several:

- conventional approaches to certification, such as measures of code coverage used to gauge the thoroughness of testing, do not take into account those models. In conventional terms, the models would look like data, and typical code coverage metrics would fail to capture the need for coverage of not only the reasoning engine’s code, but also the data encoded within the models.
- the overall ISHM system’s behavior might be sensitive to small changes in either of the reasoning engine itself (e.g., a small change to a heuristic might lead to drastic changes in performance) or the models (a small change to a model might push the reasoning engine into previously unexplored regimes of behavior) – it is hard to extrapolate (and therefore hard to know how to test) when and how these small changes will affect ISHM behavior
- the performance (run time, memory consumption, cpu utilization) of reasoning engines themselves, because of their heuristic nature, is hard to guarantee. If they are operating close to the computational “cliff” (where performance degrades rapidly as the problem complexity increases only slightly), they will exhibit occasional wild fluctuations from “normal” – for many runs it may perform within expected bounds, but once in a while, the performance is extremely poor (slow, huge memory usage, ...).

ISHM must correctly report failure conditions, and, importantly, must avoid “false alarms”. Both of these require that ISHM take as input uncertain data, and yield information and decisions with high(er) certainty. E.g., ISHM needs to distinguish engine failure from failure of the sensor(s) monitoring the engine’s health (those sensors are fallible devices, and may themselves fail). The ISHM algorithms (and implementation thereof) that perform its certainty-increasing process must be extremely reliable, since they will be in continuous operation.

Lastly, many of the systems whose health ISHM is to manage will themselves contain software. In such cases ISHM may be expected to be cognizant of the health of those systems’ software. However, software “failure” does not completely parallel hardware “failure” (software doesn’t “wear out”, rather, during operation a latent defect – “bug” – in the software may become manifest in the particular execution path it follows), so it is much less understood whether ISHM techniques can accommodate failure modes that have their origin in latent software defects (predict them for prognosis purposes, diagnose them once they have occurred, and in either case know what to do in response).

3 OPPORTUNITIES FOR EMERGING V&V TECHNIQUES SUITED TO ISHM

The unusual nature of ISHM software raises both challenges for V&V and certification (outlined in the previous section) and *opportunities* to make use of some new

and emerging V&V techniques that offer the promise of overcoming some of those key challenges. This section describes the origins of those opportunities, and gives some representative examples of emerging V&V techniques.

We have remarked on the prevalence of model-based reasoning within many ISHM architectures. In order that ISHM can perform its reasoning (e.g., diagnose the cause of a fault from a set of symptoms), those models are designed to be machine-manipulable – by the ISHM reasoning engine itself. Fortuitously, there has been an emergence of V&V techniques suited to various forms of analysis of such models.

Many of the emerging V&V techniques do their own reasoning – for V&V purposes – over the same kinds of models that ISHM utilizes. The adoption of those V&V techniques in *traditional* software settings has always been impeded by the need to construct such models by hand, from the various forms of system documentation intended for human, but not computer, perusal (e.g., requirements stated in paragraphs of English). This has made them costly and time-consuming to use, and as a result their application has, in practice, been limited to only the most critical core elements of software and system designs (for an in-depth discussion, see [Rushby, 1993]). A representative example drawn from the spacecraft fault protection domain is [Schneider et al, 1998]’s use of “model checking” was applied to the checkpoint and rollback scheme of a dually redundant spacecraft controller. In contrast, in ISHM such models are machine manipulable, and available early in the lifecycle.

Another source of opportunity offered by model-based reasoning is that the reasoning software can yield both its result (e.g., a diagnosis), and the chain of reasoning that led to that result. That chain of reasoning provides opportunities for cross-checking – not only checking that the result is correct, but also that it is correct for the right reasons (e.g., all the appropriate information was taking into account when arriving at its conclusion).

Lastly, we observe that the typical architecture of model-based ISHM divides the system into a generic, and therefore reusable, reasoning engine, and system-specific models. The reasoning engine itself is a non-trivial piece of software, and so the correctness of its implementation needs to be checked. However, since it will be reused from application to application, the effort it takes to check that implementation can be amortized over those multiple applications.

The table below lists a sampling of the applicability of emerging V&V techniques to ISHM V&V needs. Please note that this is not intended to be a comprehensive survey of the field.

ISHM problem addressed	V&V technology
Assuring the extremely high reliability of ISHM’s core algorithms (e.g., voting schemes), supportive implementations, etc.	“Formal methods” (a.k.a. “analytic verification”): theorem proving and/or model checking
Verifying key properties of ISHM models and systems, and (to some extent) of their implementations	“Formal methods” (a.k.a. “analytic verification”) : theorem proving and/or model checking
Achieving highly confidence in the correctness of the implementation (coding)	Program synthesis

ISHM problem addressed	V&V technology
step by “correct -by-construction” code synthesis methods possibly coupled with proof checking (significantly aided by information left by the construction process)	
Assuring with extremely high confidence the absence of software implementation defects in ISHM software itself	Static analysis
Eliciting software and software-system failure modes of ISHM'd systems	Risk analysis methods (SMFEA, SFTA)
Assessing ISHM's reliability for the software part of systems.	Probabilistic Risk Assessment (PRA) for <i>software</i> systems
Recognizing symptoms of software defects and, to some extent, the <i>potential</i> for software defects (i.e., expanding the confidence that can be gained from a single test run, so in part helping to counter the explosion of possible executions that ISHM systems manifest).	Run-time monitoring
Assuring that the reasoning engine not only reached the right conclusion, but did so for the right reason (i.e., expanding the confidence that can be gained from a single test run, so in part helping to counter the explosion of possible executions that ISHM systems manifest).	Testing adapted to model-based systems
Improving (by inspection, review, etc) the quality of the “models” on which ISHM operates	Software development practices adapted appropriately for models of model-based software

4 V&V FOR CONSIDERATIONS FOR ISHM SENSORS AND AVIONICS

Integrated System Health Management (ISHM) relies on information derived from sensors by avionics (signal conditioning, data conversion and data processing hardware) to assess the state of the system. The performance of the ISHM system is dependent upon the fault coverage by the sensors embedded in the space vehicle. The quality of data from the sensors and the overall reliability of the hardware of the ISHM system are critical to ISHM performance. In addition to meeting functional requirements, the ISHM system must be certified to operate reliably in the space environment. Environmental requirements for certification will include launch vehicle dynamics (vibration, shock and acoustic), thermal environments (orbital and re-entry), electromagnetic compatibility, and radiation (primarily single-event effects, since man-rated vehicles typically minimize exposure to trapped radiation environments).

ISHM Hardware Certification

Spaceflight hardware certification ensures that the ISHM hardware meets specified design, manufacturing, life, and environmental requirements. For new hardware, preliminary and critical design reviews are conducted to ensure that all basic safety, reliability, and quality assurance requirements are met. A certification requirements document identifies and defines the induced and natural environments and methodologies used in the certification approach. Certification consists of qualification testing and analysis at the highest practical level of assembly, installation, or system. Qualification testing must consider functional, environmental, and life requirements. Certification may be achieved by analysis when testing is not necessary, feasible, or cost-effective.

Spaceflight Hardware V&V

Spaceflight hardware is generally developed via a requirements-driven process where the capabilities, performance specifications and physical characteristics are developed within the constraints of mission resource allocations. High-level (system) requirements are translated into lower-level requirements, ultimately resulting in specifications that become the basis for hardware design. Validation is performed via thorough requirements traces (upward and downward) to ensure correct requirements are established at all levels. Throughout the hardware development, the compliance of the hardware design with the requirements is verified early in design reviews and later, in the hardware test program. Often a matrix is generated and maintained to track the verification of the hardware against requirements on that hardware. A performance baseline for verification of hardware functionality is established prior to subjecting the hardware to a battery of environmental tests. Abbreviated functional testing is frequently performed during the series of environmental tests (i.e. between vibration tests on each axis of the hardware). Testing of payload or subsystem avionics hardware is generally performed at the electronics box level prior to delivery to the space vehicle for integration.

System-level functional testing is often performed with engineering model or prototype subsystem hardware early in the integration phase. Testbeds are frequently employed to develop system-level functionality (command and data handling subsystems). Flight hardware can be verified in testbeds that have the appropriate interfaces and hardware protection. During integration of the space vehicle, flight hardware subsystems are typically connected to the space vehicle power and data systems via a “safe-to-mate” verification procedure. Pin-level verification of the interfaces are performed through “break-out box” equipment until the unit being integrated has been powered and proper communication is verified. Only then unit is directly mated to the flight system connectors. After all of the flight hardware has been integrated, system-level testing is completed. Robotic spacecraft typically undergo system-level environmental testing (EMI/EMC, vibration, acoustic, system thermal-vacuum tests) to verify system performance in simulated launch and space environments.

Sensor Data V&V

Due to the potentially large number of sensors, many of which are exposed to harsh environments, the ISHM system must be tolerant of sensor faults. The processes for

the selection, qualification and installation of sensors are important factors for minimizing sensor faults. An ISHM system should be able to validate sensor readings and diagnose sensor faults in real-time. The area of Sensor Failure Detection, Isolation and Accommodation (SFDIA) is being addressed by a several approaches.

There are two conceptually different approaches to the SFDIA problem: physical and analytical redundancy. Traditional flight control systems deploy triple or quadruple physical redundancy in their network of sensors to achieve the level of reliability necessary for manned spacecraft or aircraft certification. Physical redundancy SFDIA techniques are based on voting and mid-value selection schemes. It is clear that there are penalties such as mass, power, volume, and cost associated with a physical redundancy approach to the SFDIA problem.

Most of the current research activities on SFDIA focus on the use of analytical redundancy techniques. A partial list of analytical SFDIA techniques includes Generalized Likelihood Ratio (GLR); Multiple Model (MMKF), Extended, and Iterative Extended Kalman Filtering (MMKF, EKF and IEKF); Sequential Probability Likelihood Ratio Test (SPLRT), and Generalized Likelihood Test/Maximum Likelihood Detector (GLT/MLD). These techniques feature a continuous monitoring of the measurements from the sensors. At nominal conditions, these signals follow some known patterns with a certain degree of uncertainty due to the presence of system and measurement noise. However, when sensor failure occurs, the observable outputs deviate from the predicted values calculated on-line or off-line from estimation scheme generating a residual. A sensor failure can be declared when the associated residual exceeds, for a single or for multiple, time instants, a certain numerical threshold.

Analytical redundancy and Bayesian decision theory was combined to produce a sensor validation system concept for real-time monitoring of Space Shuttle Main Engine telemetry.¹ The validation system, as illustrated in the block diagram below (Figure 1), was implemented in Ada and hosted on a Boeing X-33 prototype flight computer (R3000 at 25 MHz). SSME telemetry was played back at real-time rate through the system at the Marshall Avionics System Testbed (MAST). Data from 50 SSME flight firings were processed at real-time rates and 3 sensor failures were correctly identified.

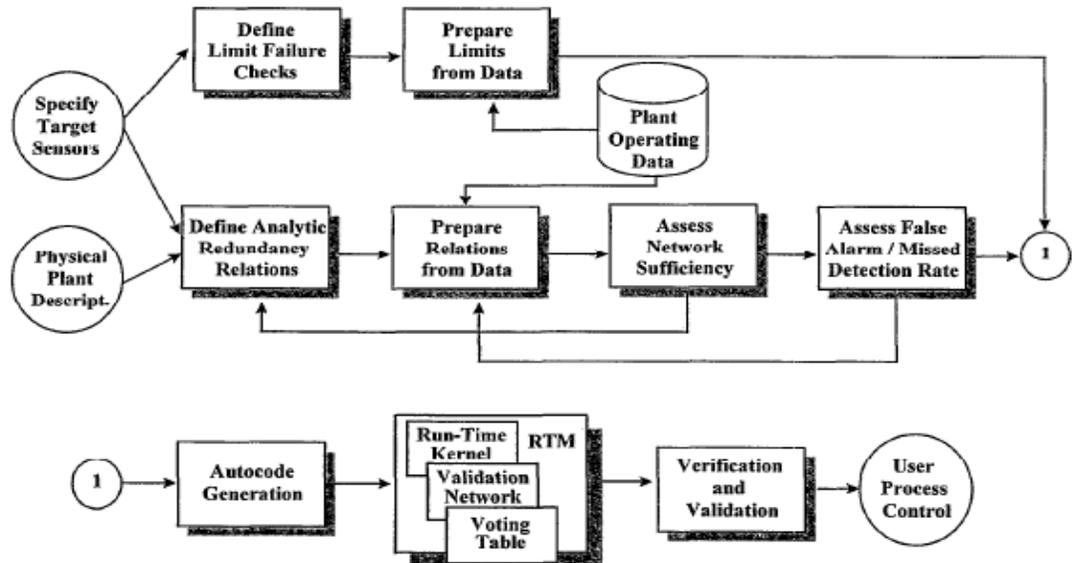


Figure 1. Block diagram for the real-time SSME sensor data validation concept developed by Bickford, et al.

More recently, neural network (NN) approaches to sensor data validation have been developed. As an example, data from a Boeing 737 was processed via a NN-based on-line learning scheme.ⁱⁱ The Extended Back Propagation (EBP) algorithm was used by the authors for the on-line learning. The algorithm was selected for its performance in terms of learning speed, convergence time, and stability when compared to the conventional Back Propagation (BP) algorithm. The SFDIA scheme is illustrated in the block diagram shown in Figure 2. It consists of a main NN (MNN) and a set of 'n' decentralized NNs (DNNs), where 'n' is the number of the sensors in the flight control system for which a SFDIA is desired. The outputs of the MNN replicate, through on-line prediction, the actual measurements from the 'n' sensors with one time instant delay, that is a prediction of the state at time 'k' using measurements from 'k-1' to 'k-p' to be compared with the actual measurement at time 'k'. In their study, the authors processed flight data obtained from about 10,000 seconds of B737 flight recorder data to train the MNN and DNNs. Simulated sensor failures were injected to test the response of the NN. They were able to demonstrate rapid on-line learning and proper identification of a variety of sensor failures both hard (complete sensor signal loss) and soft (drift) and to have the failed sensor data accommodated by the physical model adapted by the on-line learning process.

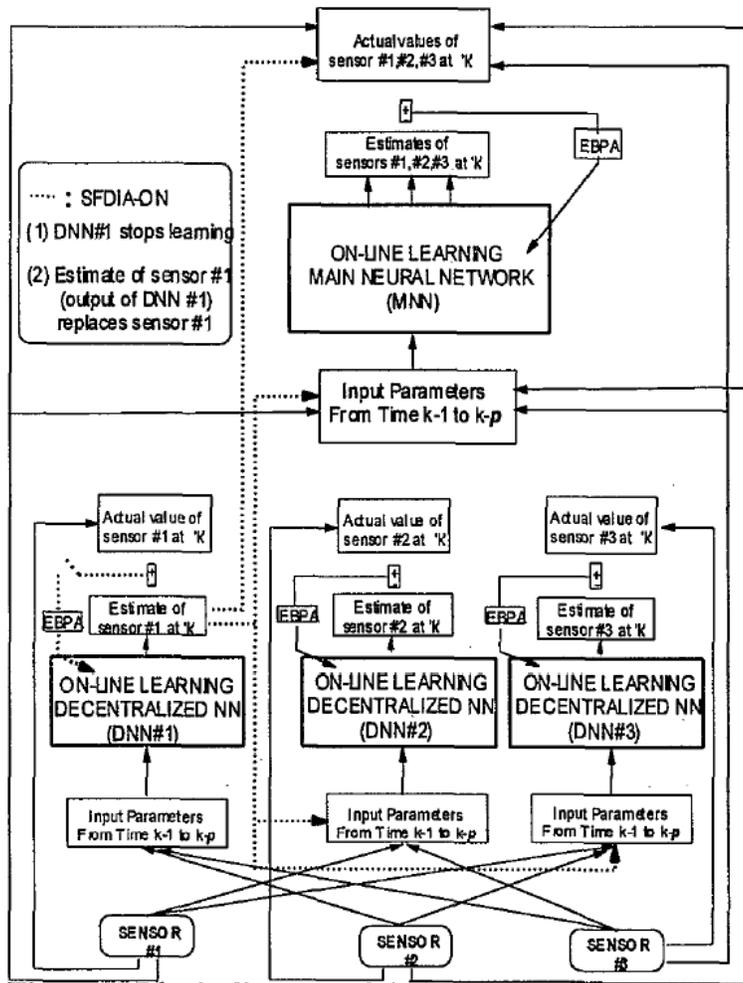


Figure 2. Block diagram of NN SFDIA scheme showing result of a failure in Sensor #1.

5 SUMMARY

It is apparent that a combination of *several* V&V approaches will be required for ISHM. For example, depending on the ISHM architecture, traditional testing approaches may be adequate and appropriate for some layers of ISHM functionality, whereas the use of AI techniques such as reasoning under uncertainty and mission planning (and re-planning) have characteristics that set them apart and challenge standard V&V techniques. Most notably, AI techniques based on explicit use of model-based reasoning exhibit algorithmic and implementation intricacies (within their AI reasoners themselves) on a par with other complex software systems, but in addition the behaviors they may exhibit during operation depend critically on the models themselves (elements that traditional V&V has not had to deal with). Fortunately, the additional V&V challenges their model-based nature gives rise to are balanced by the enhanced opportunities to apply certain V&V techniques, especially those based on analytic methods.

The function of ISHM is to increase the reliability of the system whose health it manages. To do this, ISHM must, in the vast majority of cases, correctly ascertain the status of the system, despite the fallibility of both the system itself and also the sensors that monitor the status of the system. Thus the reliability of the ISHM's core functionality (e.g., its voting algorithms that adjudicate among multiple – some possibly erroneous – readings from multiple sensors) must be among the most reliable software systems on the entire vehicle. These considerations, coupled with the non-traditional architecture that many ISHM systems employ (notably model-based reasoning), call into question both the feasibility and adequacy of existing standards, practices, etc., for V&V and certification if they are to encompass ISHM. Overall, therefore it is necessary to modify V&V and certification requirements to permit its use, to find the right mix of V&V and certification to match the architecture of the vehicle and ISHM system, and to guide the maturation of emerging V&V techniques to become capable of this function.

6 ACKNOWLEDGMENT

The research described in this paper was carried out by NASA Ames Research Center, NASA Stennis Space Center, and the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space administration. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not constitute or imply its endorsement by the United States Government or the Jet Propulsion Laboratory, California Institute of Technology.

7 REFERENCES

(Schad) Johnson, L., "DO-178B, "Software Considerations in Airborne Systems and Equipment Certification",” October 1998, available at: <http://www.stsc.hill.af.mil/crosstalk/1998/10/schad.asp>

[Rushby, 1993] "Formal Methods and the Certification of Critical Systems” Technical Report CSL-93-7, Dec 1993, SRI International, Menlo Park, CA.

[Schneider et al, 1998] Schneider, F., Easterbrook, S.M., Callahan, J.R. & Holzmann, G.J. "Validating requirements for fault tolerant systems using model checking", 3rd Int. Conf. on Requirements Engineering, 6-10 Apr 1998, pp 4-13.

ⁱ Bickford, R.L., Bickmore, T.W., Meyer, C.M., Zakrajsek, J.F., "Real-Time Sensor Data Validation for Space Shuttle Main Engine Telemetry Monitoring", AIAA-1999-2531, 35th AIAA/ASME/SAE/ASEE Joint Propulsion Conference and Exhibit, Los Angeles, California, 20-24 June 1999.

ⁱⁱ Napolitano, M., An, Y., Seanor, B., Pispitsos, S., and Martinelli, D., "Application of a Neural Sensor Validation Scheme to Actual Boeing 737 Flight Data", AIAA-1999-4236, AIAA Guidance, Navigation, and Control Conference and Exhibit, Portland, OR, Aug. 9-11, 1999