Run-time Environment and Design Application for Polymorphous Technology Verification and Validation (READAPT V&V)

DARPA Polymorphous Computing Architectures (PCA)
Contract: F33615-00-C-1887

Avionics
Verification and Validation Requirements Specification Document
Version 1

Prepared by:
Lockheed Martin
Advanced Technology Laboratories
1 Federal Street; Bldg A&E 3W
Camden, NJ 08102
(856) 338-2261
jetzl@atl.lmco.com
1 Introduction
The complexity of avionics functionality and software are increasing commensurately with the advancement of processing-resource capability. Because complexity is related to the degree of interaction between pieces of any kind, and generally a nonlinear phenomenon, it will increase at a rate conservatively greater than Moore’s law; that is, it will increase greater than twofold over approximate 18-month periods. Now with the coming of Polymorphic Computing Architectures (PCA), system complexity will increase at an even higher rate.

The reason for this increase is the added dimensionality of computations and procedures becoming instantiated “just-in-time” for their execution with “just the right” resources. With this notion, capabilities can be added with incremental changes in computational resources that can potentially exceed those realized with today’s conventional architectures. However, this is only true if the overhead incurred in PCA implementation is managed appropriately.

Because of increases in complexity, the Verification and Validation (V&V) time of avionics functions is increasing disproportionately faster than the time of other phases of the system-development process. PCA developers must understand the current and emerging approaches to avionics V&V.

With this in mind, we will first identify the V&V requirements for avionics architectures. The objective is to identify the V&V process in complex avionics systems with mission- and safety-critical requirements.

We start by clarifying the objectives of the V&V process, and what are the aspects that are specifically involved in verification. Our focus is primarily on embedded software and associated processing hardware because these are the elements of avionics that are relevant to PCA.

Next we discuss briefly the peculiarities of the different types of avionics software, classified by the flight- and mission-critical domains. These allow us to identify and summarize current requirements and approaches for avionics V&V and then subsequently critique and identify shortfalls in these methods.

Industry, universities and government agencies have recognized the inadequacies of the software-development process and are inventing and evolving methods and tools to enhance the process, including those related to V&V. The developmental steps in the emerging process are becoming intimately related and coupled, almost seamlessly bounded. While not exhaustive, we highlight and discuss some particular efforts of interest to the Lockheed Martin team.

We conclude with a discussion and summary of the development approach we are following on incipient programs.

2 Verification and Validation Objectives
This section contains general requirements for V&V of avionics software. As such, clauses that are requirements contain the word ‘shall’.

2.1 Validation
Actions to determine if the system or software requirements and the final, as-built product fulfill their intended use are considered to be validation.

Validation should take place very early in the product development cycle. In many military procurement contracts, there is a development phase devoted mainly to the validation process. This takes place prior to full-scale development or manufacturing phases. The validation phase is concerned with finalizing the customer requirements.

Validation testing usually involves building rapid prototypes that ‘mimic’ the system to be built, incorporating any constraints associated with the operation in the behavioral models. If the system under development is large, then the validation process may involve only part-task simulations and
not necessarily a full-mission simulation. The customer is able to ‘mimic’ the system in a simulated environment that responds in a characteristic manner. It is unlikely that the system will contain operational hardware and software.

In a PCA context, system validation might involve arriving at constraints on the architecture and operating system related to size, weight, energy, power and time.

2.2 Verification

Any actions to determine if the software products of an activity fulfill requirements or conditions established for these products in previous activities are considered to be verification. Verification should be integrated with the process it supports, such as supply or development, and it may include activities such as analysis, review, and test.

This activity shall consist of the following tasks:
- Process Verification
- Requirements Verification
- Design Verification
- Code Verification
- Integration Verification
- Documentation Verification.

These verification steps are consistent with the conventional software-development phases.

Results of the verification activities shall be documented and be made available to the acquirer and other involved organizations in accordance with the appropriate processes and practices for the above verification activities. Problems and non-conformances detected by the verification effort shall be dispositioned in accordance with a particular Problem Resolution Practice. All problems and non-conformances shall be resolved.

A brief description of what transpires during each of the verification phases follows.

2.2.1 Process Verification

The process shall be verified in accordance with Software Quality Assurance Practices. Typical evaluation criteria include:
- Project-planning requirements are adequate and timely.
- Processes selected for the project are adequate, implemented, being executed as planned, and compliant with the contract.
- The standards, procedures, and environments for the project's processes are adequate.
- The project is staffed and personnel trained as required by the contract.

2.2.2 Requirements Verification

The requirements shall be verified in accordance with the activities in the Development Process and the Software Product Evaluation Practice. Typical evaluation criteria include:
- The system requirements are consistent, feasible, and testable.
- The system requirements have been appropriately allocated to hardware items, software items, and manual operations according to design criteria.
- The software requirements are consistent, feasible, testable, and accurately reflect system requirements.

2.2.3 Design Verification

The design shall be verified in accordance with the activities in the Development Process and the Software Product Evaluation Practice. Typical evaluation criteria include:
- The design is correct and consistent with and traceable to requirements.
- The design implements proper sequence of events, inputs, output, interface, logic flow, allocation of timing and sizing budgets, and error detection, isolation, and recovery.
- The selected design can be derived from requirements.
2.2.4 Code Verification

The code shall be verified in accordance with the activities in the Development Process and the Software Product Evaluation Practice. Typical evaluation criteria include:

- The code is traceable to design and requirements, testable, correct, and compliant with requirements and coding standards.
- The code implements proper event sequence, consistent interfaces, correct data and control flow, completeness, appropriate allocation, timing and sizing budgets, and error detection, isolation, and recovery.
- Selected code can be derived from the design or requirements.

2.2.5 Integration Verification

The integration shall be verified in accordance with the activities in the Development Process and the Software Product Evaluation Practice. Typical evaluation criteria include:

- The software components and units of each software item have been completely and correctly integrated into the software item.
- The hardware items, software items, and manual operations of the system have been completely and correctly integrated into the system.
- The integration tasks have been performed as planned.

2.2.6 Documentation Verification

The documentation shall be verified in accordance with the Documentation Practice, and the Software Product Evaluation Practice. Typical evaluation criteria include:

- The documentation is adequate, complete, and consistent.
- Documentation preparation is timely.
- Configuration management of documents follows the procedures specified in the Software Configuration Management Practice.

As the steps above show, there is the involvement of Software Quality Assurance Practices, Software Product Evaluation Practices, and Software Configuration Management Practices. These ‘Practices’ are arbitrary in that they are tailororable within bounds to the project or product to be built. There is intimate customer involvement in the determination of what these specifics are in cases where the customer is funding the direct development of the product.

Having generally presented what the V&V requirements are for system/software development, we can clarify the process by discussing some specifics of the avionics systems we develop.

3 Air Vehicle Software Application Domains

Air-vehicle avionics and the embedded software have particulars that are distinct from other real-time software domains, such as telecommunications. Avionics form a network of generally asynchronous processing tasks and message passing, each containing their own specialized processing and input/output, reconfiguration requirements, hard and firm real-time requirements, and time-varying processing loads. It contains event- and rate-driven tasks.

There may also be strictly enforced requirements for isolation between software entities based on multi-level security considerations and flight- and mission-critical function considerations.

Avionics software can be grouped in general into two prime areas: flight- or safety-critical, and mission critical.

3.1 Flight-Critical Functions

Flight- or safety-critical functions are functions that if lost can result in a catastrophic loss of the vehicle, loss of life or damage to property in excess of a certain dollar amount. Figure 1 depicts testing philosophy and procedures for these types of functions.
In most cases—for instance in inner loop vehicular control, which is flight critical—the system enters distinct modes based on the stimuli from the environment, and it performs reasonably straightforward computations or manipulations with numerous embedded logic branches. Adding hardware redundancy in these types of subsystems increases reliability; however, with the continued increase in the reliability of processing resources, main-failure mechanisms are shifting to those that are software related.

Each mode, redundant configuration and logic pathway are tested thoroughly by varying the inputs to achieve margins of 1 failure in $10^6$ or higher. An error in the testing phase causes modification and retest, which adds weeks to the development process. Past approaches with the FAA in commercial auto-land systems perform a number of representative flight tests to collect data to verify analytical models and prove reliability of the system by complete testing of these software and hardware in the loop models.

The customer usually mandates verification requirements. For instance, DO178B provides process guidelines for safety-critical airborne-systems software. These mandates are included or referenced in the "Practices" documents mentioned above.

![Safety Critical (Certified)](image)

**Testing Philosophy**
- Full Code Path Coverage
- Both Analytical and Empirical Verification of Computations
- Rigorous Test of Redundancy and Fault Tolerance Infrastructure
- Achieve Safety Assurance in Presence of Hardware Induced Computational Errors

**Typical Procedures**
- Analytical Verification of Computations and Comparison to Actual Outputs
- Analysis of Computational Boundary Conditions
- Analytical and Experimental Verification of Redundancy and Fault Tolerance Infrastructure (Voting Scheme, Isolation Mechanisms)
- Formal Methods Used in Some Applications

**Example Applications**
- Inner Loop Flight Control
- Nuclear Stores Management
- Secure, Encrypted Information Systems

![Figure 1. Safety Critical Function Verification](image)

### 3.2 Mission Functions

Examples of mission functions are tactical sensor functions, such as radar, pilot-vehicle interface through displays and controls, and weapon-delivery computations. The integration of mission functions is also a mission-critical function. Figure 2 shows mission-critical function testing philosophies and procedures. Mission functions can be mission-critical or not. A loss in a mission-critical function usually causes a return to base to service the problem. Losses of other mission functions are not necessarily mission-critical, but they may affect the performance during the mission. For instance, loss of a secondary display may not be mission-critical because a pilot can call-up these functions on other displays. While not being mission-critical, there is an attendant increase in the pilot's workload.

Mission software is generally characterized by a large number of logic branches that interact through a large number of functional components, generally causing an almost uncountable number of threads.
through the software. The software is primarily event driven, although routines can be scheduled based on rate- or event-driven scheduling schemes. For example, note the interaction with a pilot. There is a large number of possible sequences that can be exercised as a result of display and control-switch actions. Even if the pilot exercises the same sequence, the time between actions will be different and the operational environment will not be the same.

This makes exhaustive testing cost prohibitive, so attempts at requirements coverage are usually attacked in the test phase by exercising the system with representative scenarios to ensure proper performance. Somewhat exhaustive testing at the software-unit level precedes this. There are likewise severe impacts to schedule when defects in the mission software are discovered in the testing phases.

### Mission Critical

- **Testing Philosophy**
  - Exhaustive Testing Often Impractical Due to Complexity
  - Assure Correct Application Behavior in Presence of Known Pathological Conditions
  - Rigorously Verify Manual Override Capabilities along with Infrastructure that Reports Faults to the Operator

- **Typical Procedures**
  - Manual Testing at Unit, CSC, CSCI, and System Levels (Helps Manage Complexity)
  - Manual Generation and Execution of Domain-Specific Tests (Known Pathological Conditions)
  - Manual Check of Code Paths
  - Higher Levels of Testing (CSCI and System) use Scenario-Based Approach to Ensure Correctness of the Most Important Functions

- **Example Applications**
  - Operator Interface (Displays and Controls)
  - Sensing System Control Applications
  - Decision Aiding Applications (Route Planning)

---

**Figure 2. Mission-Critical Function Verification**

Flight-critical and mission software have two very distinct rudimentary requirements. As such, the requirements for software development tools are also different (Ref. section 8).

The advent of Unmanned Air Vehicles (UAV) has blurred that which has been traditionally separated into flight and mission-critical systems, raising the level of complexity in systems to be tested exhaustively.

Autonomy functions that allow the vehicle to make decisions based on embedded meta-models of the vehicle itself may lead to unpredictable behavior when coupled to the external environment. This is an additional complication for the verification process.

The reliability of UAVs must be increased to realize their full potential. Thus, the cost and impact of V&V on developmental schedules will increase even further unless there are significant changes to the testing paradigm.

The system V&V complexity increases even further with the use of PCAs.
4 Conventional Approaches to Avionics Software V&V

Figure 3. Current Software Development Process shows the conventional software-development process with the methods in which transitions occur from one development state or phase to the next. This process prevails on many aerospace projects or programs. It is less common now in industries such as telecommunications and automotive although still exercised.

The phases correspond generally to the verification steps in section 2.2 above.

System requirements are derived from validated user requirements. The user requirements are allocated to various software and hardware functions. The output of this phase is called a Systems Requirements Document (SRD). Analysts study it and then develop system or subsystem models. Structured analysis and associated tools have mostly given way to an object-oriented analysis with some appropriate mix of CASE tools. Analytical models have been developed in the past using Fortran or C or, in the case of flight control personnel, custom-developed languages. The output of this analysis phase generally results in an SRD and an Interface Control Document. The system requirements phase and the analysis phase conventionally involve systems engineers.

Software engineers interpret these documents to derive and document top-level and detailed designs. This phase partitions the software and assigns these packages to software developers. With these, flight code is then generated using relatively ubiquitous tools for code development. These tools contain resources to create, edit, compile, link libraries and execute portions of the developed code. The code can be downloaded to a hardware emulator or the target hardware itself to be unit tested.

Meanwhile, test input cases with corresponding outputs are usually generated from the analytical models developed in the analysis phase, documented, and passed to the test engineer. The test engineer transforms the data into manageable scripts for unit and integration testing.
5 Critique and Shortcomings of Past Approaches

The percentage of software development cost for V&V varies from 40 to 70 percent. Many inadequacies in the software development process needlessly contribute to development cost. The system specifications are written in English, and the interpretation of requirements is often subjective since any spoken language is not exact.

Part of the ambiguity is removed when the requirements are converted to analysis models, but not all of it. The manual conversion to models is time consuming as is the subsequent manual conversions to design and coding. Embedded in each of these conversions is the increased potential for error that accompanies large manual tasks.

Furthermore, the writing of test scripts is prone to error and open to debugging. Errors found in unit and integration testing must be resolved in accordance with specified practices. (Ref. section 2.) This usually means a determination as to whether or not the problem is real. If so, then there must be a resolution—using analysis, a re-design, re-coding, and retesting. Some efficiency in this process is realized by upgrading problems in blocks; however, there is still significant labor expended in the rework loops found in Figure 3. Current Software Development Process.

6 Identification of Technologies Enhancing the V&V Process

From the discussion above and Figure 3, the development process could be more efficient with a reduction in manual intervention. There has been a focus over several decades in automating most of the transition steps. This focus has also included the development of analysis, design and testing tools that reduce errors that tend to propagate to later phases. Perhaps the additional cost and time spent prior to testing is warranted because it will reduce the time to test and rework.

If a break occurs in the formal specification and production phases, it becomes difficult to establish an overall productivity gain, despite the extra cost. A good example is the step between manual coding and testing the code to ensure that its behavior complies with the design specifications.

If downstream production of the software is guaranteed to comply with the specifications through automated CASE tools and automatic code generation (continuous process), the gain is threefold: fewer specification errors, lower software production costs, and reduced software testing costs, provided that the tests remain valid for the generated code. This occurs if the selected tool can guarantee behavioral equivalence between the specification simulator and execution of the generated code.

Formal Methods is a phrase describing approaches that guarantee this form of rigor. Formal methods can generally be divided into two main areas: Specification and Verification, which contains the approaches of Model Checking and Theorem Proving.

Formal Methods have been slow to infiltrate aerospace because the techniques are relatively obscure to the engineers involved in the development process, and there is a lack of integration of these methods with tools that are familiar to the developer. The systems that aerospace produces are large and complex and take significantly longer to develop compared to commercial ones. This creates a corresponding inertia that is resistant to change. However recent trends (see below) suggest that methods are poised to have a greater influence on this market.

Specification when used in the formal methods sense refers to the analysis step in Figure 3. It refers to "Model" specification and not the Systems Specification normally contained in the SRD in the figure. It relies on using unambiguous language to encapsulate the requirements derived from phase 1 in the figure. These methods have generally been most successful in modeling the behavior of the system.

This capability has great potential when integrated with the system-development process. Avionics software can generally be decomposed into reactive portions and computational portions — hybrid models consisting of discrete transitions between states, where the computational pieces are invoked. The reactive portion of the software only represents 30 to 40 percent of the number of lines of source code,
but it accounts for 70 percent of specification or coding errors. Due to its combinatory nature, event or reactive processing has been considered difficult to master. Numerous formalisms and tools have been designed to help solve this problem; the most recent and promising is StateCharts because tool developers accept the tool and methodology, e.g., iLogix’s Statemate or Rational Rose-RT and toolkits for dynamic systems such as The Maths Works’ Matlab.

A model checker investigates every possible behavior of the system for all initial conditions and input signals, while a simulator generates only one trajectory for a particular initial condition and input signal. In the area of verification, model-checking is beginning to eliminate shortcomings of model explosion though the use of binary-decision diagrams. Techniques have an added advantage: while exhaustive checking of system state values is occurring, counterexamples can also be produced. When coupled with model-slicing techniques (tracing pathways involved in influencing a given variable), this capability is useful in the debugging process. See http://www.ece.cmu.edu/~krogh/ford/sf2smvindex.html for a downloadable copy of the sf2smv (State Flow to Symbolic Model Verifier) program that can be used as a utility in Matlab.

The above-developed techniques are most useful for primarily reactive portions of software. Extensions to a complete hybrid system are still emerging and still cumbersome for the systems engineer; however, these extensions, like the Checkmate module, are also being integrated with commercial toolsets, such as Matlab.\(^2\,3\)

Automatically generating test vectors and automating the test environment are areas where large payoffs can occur. These significantly reduce the time generating test scripts and ensure completeness in test coverage. See Reference 4 for an integrated approach with aerospace applicability.

7 Ongoing Technology Thrusts and Program Efforts

We are cultivating a variant development paradigm (for mission-avionics applications) that uses automation-assisted V&V technologies to address these issues. Under this approach, executable UML-based CASE models (object and state views) are built and continuously checked and refined during development. The overall goal is to have an executable, object-oriented modeling environment coupled with the rigor, coverage, and automated V&V support previously associated with formal methods. There are several external parties investing in automation-assisted checking of CASE models built with mainstream commercial tools:

7.1 DARPA/ITO MoBIES

This project is funding approaches to add model-checking capabilities to UML and Matlab representations.

7.2 AFRL/IFSC FACIT

This project will extend MoBIES and commercial CASE checking capabilities and develop practices and procedures for large-scale project deployment. Lockheed Martin Aerospace (LM Aero) and Lockheed Martin Advanced Technology Laboratories are teammates under D.O. #3 of the FACIT contract.

7.3 LM Aero - I-logix Corporation Collaboration

A recently formalized LM Aero / I-logix corporation collaboration will allow us to influence the capabilities and design of the emerging test-automation suite that is being built for Rhapsody (the primary mission applications CASE tool on JSF and an advanced F-16 variant). I-logix is investing heavily in formal methods technologies to enhance its current product lines.

7.4 AFRL/IFTA - LM Aero Model-Based Verification

This is a pending AFRL/IFTA / LM Aero contractual link with an ongoing Software Engineering Institute (SEI) project called Model-Based Verification. It will leverage SEI experience with formal modeling and assist the above three efforts in upgrading the current V&V state of practice.
8 Emerging State of the Avionics Software V&V Process

From the discussion above and figure 4, V&V is embedded in every step of the system- and software-development cycle. It is a time-consuming, full-time effort. It cannot occur after the fact. Figure 4 shows that in the normal software-development process, requirements generation immediately spawns the development of the test cases, and test engineering activities, paramount in the V&V process.

The test and V&V phases take 40 to 70 percent of the software-development time and cost. The emerging push is for additional work up-front to reduce the rework that leads to large chunks of effort and time.

Software for reuse and commercial developmental tools for the layman is key to this realization. Figure 5 shows a sample set of tools that correspond to the steps in Figure 4 for the integrated engineering-application environment.

PCAs will complicate this V&V process. The initial introduction of PCA technologies into the aerospace arena will likely be in mission-critical applications. The aviation industry is notoriously slow to introduce unproven technologies into its already meticulous V&V process, thus making mission-critical applications the most likely initial vehicle.

The introduction of PCA can at best lead to a multiplicative effect in the V&V effort. If the system of morphware and hardware are not introduced in a proper manner, then this introduction can lead to a factorial impact on the V&V process.

For proper and streamlined infusion of PCA morphware and hardware, the technology must be developed much like previous operating systems. These are independent packages with appropriate interfaces and “device drivers”–like implementations that involve specifics of actual hardware. This includes software that is not intimately integrated with the applications software. The interfaces are clean and follow certain well-defined standards. This means that the technologies have evolved to a maturity that can be classified and typed accordingly.

The PCA developer can then independently certify the PCA technologies. With the certification process, the morphware/hardware developer can provide artifacts and documentation—arrived at in the independent V&V process—to the aerospace system developer to help certify the total system. The efforts of the Morphware Forum Working Group are critical in reaching this goal and appear to be headed in just this direction.

Tools to aid in integration and requirements development for PCA components must also accompany the actual morphware/hardware. It would be preferable to include in this toolset any debug and test tools as well as a PCA development environment that could be used to tailor PCA inclusion in the total system. The most preferable mechanism would be a third-party integration or partnership with an already existing software development environment, such as MATLAB/Simulink, Green Hills and/or VxWorks.
We have presented avionics V&V methodology and requirements to provide a context for requirements development for V&V of PCA implementations into the aerospace domain. We emphasize that verification requirements are not add-ons after PCA approach(es) are developed, but they are integral to
the development of the PCA system itself. The methodology of test and verification of polymorphous systems is part of the development of these systems; that is, one can not have a PCA system without requirements for verification embedded within the PCA system itself. Realization of this will aid in the most rapid introduction of PCA into the aerospace domain.

9 References

