Architecture-Centric Virtual Integration Process Modeling & Analysis Handbook

Revision 1.1
March 2021

Prepared for
U.S. ARMY DEVELOPMENT COMMAND
AVIATION & MISSILE CENTER
Redstone Arsenal, AL

Prepared by
Adventium Labs
111 Third Avenue South, Suite 100
Minneapolis, MN 55401
Contract # W911W6-17-D-0003

This material is based upon work supported by the U.S. Army Development Command Aviation & Missile Center under Contract No. W911W6-17-D-0003. Any opinions, findings and conclusions, or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the DEVCOM AvMC.

Revision History

Revision	Date	Summary
0.1	30 October 2015	Initial draft
0.2	20 May 2016	Update prior to the JMR TD AIPD program
0.3	29 November 2017	Update incorporating JMR TD AIPD lessons learned
0.4	08 January 2018	Update prior to JMR TD Capstone BAA
0.5	02 April 2018	STPA summary added prior to JMR TD JCA Product Developer call
0.6	15 June 2018	Update prior to JMR TD Capstone MSI call
0.7	14 September 2018	Update prior to JMR TD MSAD Capstone Architect Kick-Off
0.8	March 2019	Update prior to public release
1.0	15 August 2020	Update for Future Vertical Lift Architecture Framework 2.0
1.1	30 March 2021	Incorporate JMR TD MSAD Capstone lessons learned

Changes since previous version:

- Shortened title to align with ACVIP Acquisition Management Handbook and common use
- Updated due to update of ACVIP Acquisition Management Handbook
- Updated due to update of Defense Acquisition Guidance Systems Engineering review criteria
- Added a fifth umbrella planning topic for ACVIP as a domain within MBSE and ASoT.
- Added material on Modular Open Systems Approach
- Cited System Readiness Directorate and AMACC in airworthiness subsection
- Added System Functional Review (SFR) section
- Introduced architecture viewpoint concept
- Introduced analysis specification concept
- Moved safety policy, security policy, certification subsections into same section
- Added subsection on human inspection and defect detection by modeling
- Refactored analysis specification subsections to better align with DAG SE criteria
- Added examples to illustrate updates

DISCLAIMERS:

The information in this handbook does not represent any official policies or views of the United States Army or Department of Defense. The US Army does not guarantee the accuracy or reliability of the information in this document. This document does not imply government endorsement of any product, service, or entity.

The ACVIP handbooks use the Architecture Analysis & Design Language (AADL) as the language of choice to capture architectures. Use of other Architecture Description Languages (ADLs) would require modified guidelines.

COMMENTS:

Comments on this document may be provided by sending an email to **usarmy.redstone.rdecom-amrdec.mbx.acvip@mail.mil** or calling 256-842-6600**.**

Contents

1. Executive Summary

1.1 Scope and Purpose

An Architecture-Centric Virtual Integration Process (ACVIP) addresses acquisition goals of affordability, reduced program risk, faster upgrade cycle times, and reduced risk of compromised capabilities. ACVIP supports the Department of Defense (DoD) Digital Engineering Strategy [1] and the Modular Open Systems Approach (MOSA) [2]. ACVIP addresses these goals by applying architecture-level model-based software and systems engineering methods for embedded computer systems during early development phases to reduce late-phase rework, avoid work-arounds that compromise system capabilities, and streamline certifications and future upgrades.

Studies have shown that much Engineering & Manufacturing Development (EMD) cost and schedule is due to rework that occurs in the software and systems integration and acceptance phases [3, 4]. This is often unplanned, resulting in project overruns. Sometimes the originally desired system capabilities are compromised to deal with issues found in late phases. The root causes for much of this rework can be traced back to defects in requirements and architecture and interface specifications. Expensive defects are usually not isolated inside single components; they are defects in how components are assembled and interact with each other in the overall system that are not detected until system integration testing. ACVIP provides model-based system engineering methods that are applied in early phases to avoid and detect such defects in embedded computer systems, at a point during development when they are much easier to correct.

ACVIP is a model-based dry-run of cyber systems integration early and iteratively during the development process. Virtual integration mimics the architecting and integration of a computing system using digital models. An architecture model of the overall computing system is created and iteratively elaborated. Suppliers are issued model-based specifications for components and deliver models of those components that are acceptance-tested by virtual integration. Component models are virtually integrated into the system architecture model. The resulting integrated system model is subjected to multiple analysis tools to detect defects and technical risks early, while the costs to repair or mitigate are still low. ACVIP automates the process. A large number of analyses are performed quickly and iteratively by applying multiple tools to a common system architecture model, with information increasing and uncertainty decreasing throughout the development process. ACVIP is applied to computing subsystems, which must be rapidly update-able and account for much lifecycle cost (and historically much program overrun and risk).

Because ACVIP manages technical uncertainty and risk, it mitigates program risk. Because ACVIP mimics the development process, it can give early warning of programmatic as well as technical issues, another program risk mitigator. Because ACVIP is highly automated, model integration and analysis can be iteratively performed at a rapid tempo for ongoing coordination of multiple suppliers and exploration of alternatives, resulting in more agile acquisition.

This handbook provides guidelines to engineers and engineering managers for planning and executing the ACVIP engineering tasks of an embedded computer system development project. Project planning identifies a network of development tasks, resources, and task inputs and outputs. For a project that incorporates ACVIP, the plans include engineering tasks to develop models, virtually integrate and analyze models, identify defects and their root causes, take corrective and preventative actions, support reviews and approvals, and deliver models for use by other organizations and on future projects. This handbook provides guidance for planning and executing these engineering tasks.

Readers are assumed to be familiar with the first two volumes in the ACVIP handbook series, the *ACVIP Overview* [5] and the *ACVIP Acquisition & Management Handbook* [6]. The *ACVIP Overview* provides a general introduction and describes the motivations, benefits, and basic processes and approaches of an ACVIP. The *ACVIP Acquisition & Management* handbook provides guidelines for applying ACVIP within the DoD acquisition process.

The *ACVIP Acquisition Management Handbook* provides guidance for a government audience, while this handbook provides guidance for a supplier audience. Both handbooks are written to be of use to both government and supplier, as that helps each understand the activities that the others perform. This handbook addresses contractor systems engineering and software and systems integration personnel; and government personnel who issue technical requirements, review technical results, and use delivered models and results to support planning of future projects.

Readers are assumed to be familiar with the SAE International Architecture Analysis & Design Language (AADL [7]). This is the standard semantics and modeling language recommended for Architecture-Centric Virtual Integration, and many of the guidelines in this handbook apply specifically to AADL. Concepts and terms from the AADL standard and its annexes will be used in this handbook without definition or citation. To avoid ambiguity, terms referring to specific language keywords and grammatical constructs will be written in boldface, e.g., **type** refers to an AADL type declaration, **system** refers to an AADL system declaration.

This handbook is informative rather than prescriptive. "Should" means a guideline is recommended in most cases but may not be suitable for all. "Should consider" is used for issues that arise in most projects and should be addressed in some way, but there is no single recommended way to do so.

1.2 Concepts and Terms

A system is "an aggregation of system elements and enabling system elements to achieve a given purpose or provide a needed capability. The enabling system elements provide the means for delivering a capability into service, keeping it in service, or ending its service and may include those processes or products necessary for developing, producing, testing, deploying, and sustaining the system [9]."

A model is "a representation of one or more concepts that may be realized in the physical world. Models are represented in many forms including graphical, mathematical, and physical prototypes. Typical systems engineering models may include behavioral, structural, geometric, performance, and other engineering analysis models. Model Based Systems Engineering (MBSE) is the formalized application of

modeling to support system requirements, design, analysis, verification and validation beginning in the conceptual design phase, and continuing throughout development and later life cycle phases [8][10]."

In this handbook the singular "model" refers to any collection of AADL **packages** that satisfies the syntactic requirements of the AADL standard. The term "model" has compositional and extensional semantics – multiple models may be integrated to form a system model, a system model may be decomposed into component models, and one model can be declared as an **extension** or **refinement** of another model. The term "model element" refers to any individual declaration, value, object, or grammatical clause within a model.

Architecture is "the fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution [11]." An architecture description is "a collection of products to document an architecture [11]." This handbook deals with architecture descriptions that are written in AADL, which enables analysis of architectures from multiple viewpoints to support an ACVIP.

The term "component" is used in this handbook to mean a part of a system, a "system element" in the first definition above. A system consists of an integration of components. A component may be further decomposed into subcomponents, and the term "component" is considered synonymous with "component or subsystem." AADL models of multiple components can be virtually integrated to form an AADL model of a system of components, and an AADL model of a system can be decomposed into models of its subcomponents together with declarations of how those subcomponents interact with each other.

The term "capability" is used to mean a characteristic or behavior of a system that makes that system useful for people as they carry out some activity for some purpose. The term "requirement" is used to mean information that describes functional and non-functional aspects of a system that will ensure that system provides the desired capabilities. Requirements are derived from desired capabilities as well as other information such as technology and programmatic resources and constraints.

The term "specification" is used in this handbook to denote any human-readable form of technical information about a product or its environment or its use. This may range from operational requirements specifications to rigorous detailed engineering specifications of individual components.

1.3 Handbook Outline

This handbook is divided into six sections. An ACVIP Management Plan checklist, a bibliography, and a list of acronyms are included as appendices.

Section 2: Develop ACVIP Management Plan identifies high-level goals and requirements to be satisfied and verified using ACVIP methods. It identifies the methods, tasks, artifacts, model exchanges, and schedules to accomplish this. Contractor ACVIP Management Plans for specific contracts are developed in response to customer (government) ACVIP Plans for the program. ACVIP plans define engineering tasks for multiple organizations involved in a project. Like the system engineering management plans they support, they are living plans.

Section 3: Structure Models for Delivery and Virtual Integration defines concepts and terms used to describe how models themselves are modularized for delivery and virtual integration. Models are developed, delivered, and virtually integrated to create larger models. ACVIP plans must identify and describe models in sufficient detail so that, when combined, they form an integrated model that satisfies the AADL legality rules and ACVIP Management Plan. Guidelines are provided for specific modeling patterns and AADL language features.

Section 4: Support Certification Approvals and Readiness Reviews provides guidance when modeling & analysis is applied to certification requirements. Specific guidelines for safety and cybersecurity are provided. There are additional considerations to be addressed if modeling & analysis outputs are used as evidence for certification.

Section 5: Define Model Content Needed for Analyses and Reviews provides AADL-specific guidance for capturing specific kinds of information for specific kinds of analysis. The guidelines in this section are organized using familiar major review milestones, but ACVIP is adaptable to different acquisition paths and development processes.

Section 6: Assure System Conforms to Models provides guidance to assure the to-be and as-built system conforms to its model-based specifications and the analysis results are acceptable for their intended use.

To help understand and apply these guidelines, this document includes examples and notes on related topics and rationale. Any standards, patterns, methods, tools, or other project or technical data used in examples and notes are not guidelines or recommendations. They are hypothetical stories and background to help understand and apply the guidelines.

> *Note: Notes and examples will be labeled as such, indented, and written in an italics font. Figures and tables referenced in notes and examples will also be so marked.*
>
> *Note: The text of this document includes linked cross-references to specific sections, such as the* Develop ACVIP Management Plan *section. These will be underlined to indicate they are links. Where the occasional forward link appears, effort has been made to provide some anticipatory context.*
>
> *Example: The Open Source AADL Tool Environment (OSATE) is an AADL Integrated Development Environment (IDE) that is one among several available AADL tools. The guidelines of this handbook are not specific to any particular tool, so planners and users of OSATE must consult the OSATE help files for details specific to the capabilities of that tool.*
>
> *Note: This is a living document originally developed with the support of the US Army Joint Multi-Role (JMR) Technology Demonstrator (TD) Mission Systems Architecture Demonstration (MSAD) Science & Technology (S&T) program. It is an asset in the Future Vertical Lift Architecture Framework. Regular updates will occur. Readers of this document are encouraged to submit recommendations and corrections by communicating with the listed government point-of-contact*

*for the issuing office. Comments can be provided by sending an email to: **usarmy.redstone.rdecom-amrdec.mbx.acvip@mail.mil** or calling 256-842-6600.*

2. Develop ACVIP Management Plan

This section provides guidelines for planning the ACVIP elements of an individual contracted project, which is a single program element within an overall acquisition program lifecycle. There are five umbrella planning themes that reappear in various detailed forms throughout this handbook.

- ACVIP is a model-based dry-run in advance of software and computer system integration. Model development and virtual integration plans should reflect system requirements and development plans. Models for components will be acquired from the suppliers of those components, and descriptions of models to be delivered should be given to suppliers just as they are for delivered components. Virtual integration of component models will be performed by the system integrator. Models, analysis results, and corrective and preventative actions that resulted, are reviewed at major review milestones. Problems encountered during execution of the virtual integration plans may identify risks in overall development plans, avoiding defects in project planning as well as technical defects.
- ACVIP is applied in the domain of safe, secure, real-time computer system architectures. ACVIP is planned and executed in the broader context of overall systems and multi-disciplinary engineering. ACVIP models and tools must interoperate with other models and modeling tools. Within ACVIP, a single core system architecture model is developed that is annotated and used to perform multiple automated analyses. This reuses modeling effort for multiple purposes, assures consistency between multiple analysis results, enables trade studies, and enables a rapid tempo of automated analysis iterations.
- Planning begins with the identification of purposes, goals, and requirements. Plans are then refined to meet those. For example, from a goal of reducing risk of project overrun by reducing unplanned rework, categories of integration defects that are to be avoided are identified; then analyses able to detect their root causes are identified; then descriptions of the models that must be developed to perform those analyses; then schedules for model deliveries, virtual integrations, and analyses; then activities to provide the necessary assurance the system (to-be or as-built) conforms to the model and analysis results.
- The models used for virtual integration should be part of the requirements and specifications for the system and its components. Modeling and virtual integration should be aligned with system development and integration (or more accurately, vice-versa). This provides assurance that modeling and analysis results accurately describe the to-be and as-built system.
- Models serve as a means of communication, not just as input to tools. Models should be well-structured and well-documented. Models should be accompanied by a “model users guide” that explains the purpose and how to use a model. This is particularly true for models provided as assets in a Technical Data Package (TDP) for open systems.

2.1 Acquisition Context

ACVIP plans should be tailored for each acquisition program and each project within a program. A government Program ACVIP Plan is created and managed by the Program Systems Engineer to address the overall ACVIP management approach. The supplier develops a more detailed ACVIP Management Plan that responds to the Program ACVIP Plan. This is analogous to the development of a System Engineering Plan (SEP) and a responding System Engineering Management Plan (SEMP). As with the SEP and SEMP, ACVIP plans are updated as needed throughout the program and each project. The initial ACVIP Management Plan should be completed shortly after project start, in the same time-frame as the System Engineering Management Plan.

ACVIP plans should be documented and well-integrated with overall program and project plans, but this handbook is silent about how ACVIP planning information should be captured in a specific set of documents. Some planning information may be represented using models (e.g., Business Process Modeling Notation), but the overarching plan is expected to be captured in natural language documents since much information (such as intended uses of models and rationale) is meta-data about models rather than data declared in models. The Program System Engineering Plan and individual procurements should say how these plans are to be documented and delivered. An ACVIP Management Plan document may be a deliverable, or that information may be included in other required planning documents.

The organization of portions of this handbook reflects common defense acquisition milestones (System Requirements Review, etc.). This is not intended to constrain the development processes used to meet planned milestones and deliverables. ACVIP planners should adapt these guidelines as necessary for agile, iterative, incremental, etc., development processes. The *ACVIP Acquisition Management Handbook* provides guidelines for ACVIP in the context of the adaptive acquisition framework, such as Software Acquisitions and Incremental Design Reviews [6]. The process can have a significant impact on planning for certain goals. For example, a goal of adapting to rapidly evolving requirements is more easily addressed using an agile or incremental development process, and ACVIP plans for such processes may include methods such as continuous virtual integration.

Figure 1 Notional ACVIP Project Milestones shows conventional milestones that may appear in an ACVIP Management Plan. Other review milestones may also occur, such as a mission system integration Test Readiness Review (TRR). ACVIP plans are continually refined and incrementally executed throughout the development phase. Requirements modeling and the results of virtual integration modeling and analysis will be reviewed at planned intervals. Consistency of the final model deliverables with the system deliverables will be verified at Physical Configuration Audit. Certification and readiness review schedules will depend on the project requirements identified in the Program System Engineering Plan and Program ACVIP Plan.

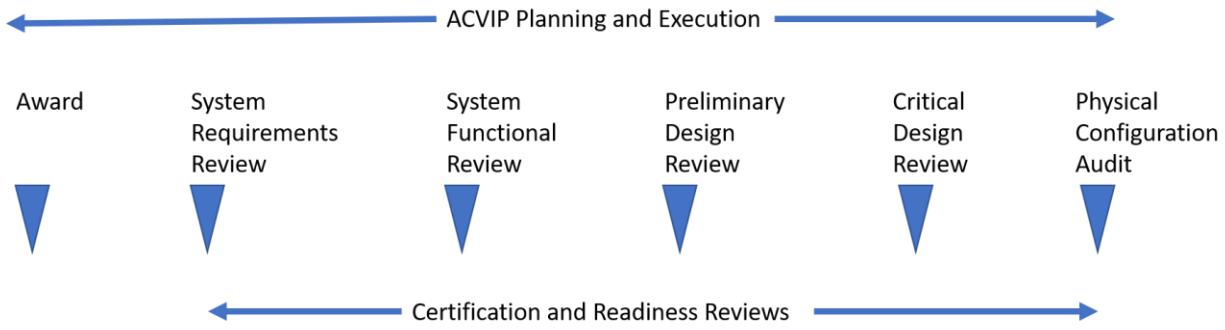

Figure 1 Notional ACVIP Project Milestones

2.2 Identify ACVIP Purposes, Goals, and Requirements

ACVIP planners should first identify goals for the planned ACVIP activities, where "goals" encompasses a variety of terms such as requirements, key business drivers, quality attributes, open systems, performance specifications, and certification approvals. Here is a list of goals that can be supported by the guidelines in this handbook. Although the basics of ACVIP are common to all these goals, the selection and tailoring of different goals for a specific project will affect the plans.

- Reduce risk of project overrun by improving early detection and correction of architectural defects that would otherwise result in significant unplanned rework during software and system integration and acceptance testing.
- Achieve Modular Open Systems Approach (MOSA) objectives by modeling key interfaces in a standard architecture modeling language with standard semantics for the domain of embedded computer systems, with interface consistency verified and upgradeability evaluated by analyses and virtual integration studies.
- Reduce technical risks by modeling and analyzing technical uncertainty; and by modeling and analysis to verify that identified technical risk mitigators can be effectively integrated if needed.
- Increase affordable capability by improving trade studies between cost versus capabilities, and by reducing the sacrifice of planned capabilities in order to stay within budget and schedule.
- Streamline certifications by using model analysis and verification results to reduce defects found in late-phase certification reviews and as evidence for certification and approval authorities.
- Reduce cost, schedule, and risk for subsequent system upgrades by delivering computer system architecture modeling & analysis assets that benefit future projects or other programs.

ACVIP plans identify component models to be developed, virtually integrated into a system model, and analyzed, in order to achieve these goals. Some models will be deliverables that must satisfy customer requirements derived from customer goals and directives.

ACVIP plans identify and describe the following.

- Goals and requirements to be met by ACVIP tasks, captured in models, and verified by analysis
- The scope, purpose, structure and content of the model(s) to be developed
- Modeling standards, libraries, and patterns to which delivered models must conform

- Virtual integration milestones, the analyses to be performed, and results to be produced
- Methods and acceptable tools to perform virtual integration activities
- Milestones, procedures, and formats for delivering models and analysis data
- Procedures for taking corrective and preventative actions based on analysis results
- Configuration management of the models and key development tools and data
- Activities to assure compliance of to-be and as-built systems with their models
- Models, analysis results, methods, and tools to support certification reviews and approvals
- Support for other related activities such as program, risk, and configuration management
- Schedules, resources, training, milestones, and performing organizations

The ACVIP Management Plan is a living plan. The initial plan should anticipate that it will be refined during project execution to improve efficiency and effectiveness of the ACVIP tasks.

> *Example: An initial ACVIP Management Plan states that an architecture model shall be developed prior to System Requirements Review (SRR) that allocates functional requirements to major software and hardware components and captures required reserves for processor loading and size/weight/power. Uncertainties in the model parameter values shall be identified, sensitivity to those parameters shall be analyzed, and high-risk elements of the model identified. The ACVIP Management Plan shall be updated after SRR with plans to further detail high-risk portions of the model and reduce uncertainty in high-sensitivity parameters prior to PDR. Preliminary analysis of processor loading and size/weight/power shall be reviewed at Preliminary Design Review (PDR) and tracked through system integration Test Readiness Review (TRR).*

Define Model Content Needed for Analyses and Reviews identifies several kinds of ACVIP analyses that may be performed at project milestones. The guidelines describe the content of AADL models needed to perform those analyses. This list is neither exhaustive nor exclusive. ACVIP plans should select guidelines from this document and tailor or extend them as needed for a specific acquisition program, family-of-systems, or system-of-systems. This handbook serves as a reference for engineers performing ACVIP tasks as much as a menu of issues to be considered by planners. The level-of-detail in the plan should be appropriate for the project, as complex as necessary, as simple as possible.

> *Example: The government is acquiring a software component that will be provided as Government Furnished Information (GFI) to other contractors on other projects. The final delivery TDP shall include an AADL model of that component that can be virtually integrated into the AADL architecture models of multiple other contractors. To accomplish this goal, the government provides as a supplement to the solicitation an AADL model that captures key interfaces and protocols of the execution environments into which the delivered model may be virtually integrated. This model has a "Your component model goes here" (analysis harness) structure. The acceptance criterion is that the delivered component model will virtually integrate into this execution environment model and then pass a specified set of interface consistency analyses.*

When describing the purpose of a model, an important distinction is model-as-specification versus model-as-description.

- If the purpose is model-as-specification, then models must be developed early and included as part of the system requirements and specifications. If the as-built system does not conform to the model, then the as-built system is defective. This is usually what is meant when engineers refer to model-based engineering.
- If the purpose is model-as-description, then the models describe existing components and environment behaviors in order to enable certain analyses. If the model does not accurately describe the as-built system or its environment, then the defect is in the model. This is usually what is meant when scientists refer to a model of a system.

This distinction may be made for models as a whole, but it is often the case that different elements in the same model have different purposes. For example, an upper bound on thread execution times or a fault rate for a processor could be requirements to be met, or they could be descriptions of a component to be reused. The plan should distinguish whether a model element is a specification or a description in order to distinguish which artifact serves as the ground truth, which is defective, and who bears the responsibility to repair the defect. This will be discussed further in Structure Models for Delivery and Virtual Integration.

2.2.1 Reduce Risk of Project Overrun

Project risk management begins with risk identification [9]. Modeling & analysis should be identified during ACVIP planning that have good early detection effectiveness for categories of defects that can cause expensive late-phase rework. The selection of analyses will depend on both the characteristics of a specific project (e.g., safety or security approvals required, prior experience with similar systems, special technical risks, cost/benefit trade of late rework reduction versus early modeling & analysis effort) and available methods and tools.

> *Example: Table 1 Example Software Requirement Defect Categories for Safety-Critical Vehicle Software summarizes two taxonomies of requirements defects that were developed during studies of defect data from several National Aeronautics and Space Administration (NASA) programs. These studies looked at requirements defects that resulted in significant late rework or significant increases in operator workload due to reduced system capabilities.*

Table 1 Example Software Requirement Defect Categories for Safety-Critical Vehicle Software

Example Software Requirement Fault Taxonomy [10]
Incomplete
Omitted/missing
Incorrect
Ambiguous
Infeasible
Inconsistent
Over-specification
Not traceable
Unachievable
Non-verifiable

Example Requirement Error Root Cause Taxonomy [11]
Requirement recognition
Requirement deployment
Interfaces not adequately identified/understood
Hardware behavior anomalies
Interface specification not documented/communicated
Interface design during testing
Lack of communication between hardware/software team
Hardware behavior not documented
Requirement not identified
Requirement not understood

Misplaced
Intentional deviation
Redundant/duplicate

Specification imprecise/unsystematic
Requirement missing from document
Incomplete document of requirement or change
Coding error persists to integration testing
Design inadequate for required function

Another source of significant rework is changes to requirements and architecture specifications made during project execution. ACVIP plans should consider the project plans and development processes put in place to respond to requirements changes. ACVIP plans should support requirements change management. Consideration should be given to ACVIP elements that can support agile acquisition processes as needed. Later sections on Identify Relations Between Models, Identify Variation Points, Configurations, and Dynamic Behaviors, and Identify Change and Configuration Management Procedures provide guidelines that support trade studies, product lines, and change management.

The root cause of expensive rework may appear in an early model not as something that clearly fails an analysis but as something that introduces unnecessary complexity and increases risk of future defects. This risk may depend on the engineering methods and processes used during subsequent development tasks. For example, the risk of later mistakes may depend on whether an engineering method will be used that is able to deal with a certain kind of complexity or selected technology in a system architecture. Models should be reviewed or analyzed to assess this category of root causes (risk due to unnecessary complexity or incompatibility with down-stream development and manufacturing processes). Preventative as well as corrective actions may result from virtual integration reviews.

> *Example: At the Preliminary Design Review (PDR), the architecture model shows that some software components use a request-response protocol for interacting with a data server. The model shows that the compute platform will comply with the ARINC 653 standard, which uses a static cyclic schedule to alternate between isolated software components. A preliminary latency analysis shows that some software components will need to send anticipatory requests a cycle before data is needed, data server applications will need to service incoming data writes before incoming data requests, and clients and data servers will need to operate at different periodic rates. The resulting software and system integration problem is potentially solvable, but it would be less complex and defect-prone for software component developers and integrators if either (1) a periodic publish-subscribe protocol was used for all software components with that execution environment scheduling protocol, or (2) an event-driven execution environment scheduling protocol were used with that software request-response protocol.*

Experience shows that the task of creating a model in a rigorous modeling language will reveal defects in other work products, such as ambiguities and missing information in a natural language document from which a model has been derived. Engineering tasks that create models also serve as a form of structured review for all the sources of information used to create that model before that model is subjected to further analysis. ACVIP plans should anticipate that corrective and preventative actions will be needed for defects detected during model development tasks.

2.2.2 Satisfy Modular Open Systems Approach Objectives

A Modular Open Systems Approach (MOSA) is required for major defense acquisition programs [2]. The Technical Data Package (TDP) of a system must be open as well as the system itself. The TDP should also use appropriate available standards. ACVIP uses a standard modeling language in the domain of safe, secure, real-time computer system architecture, with annexes to standardize alignment with other standards. This provides a standard way to specify key interfaces. Virtual integration with automated analyses can also support verification and validation of openness quality attributes.

Achieving MOSA goals may require delivery of models in standard formats that allow other parties to perform analysis, trade study, verification, and upgrade tasks using reasonably affordable and non-proprietary tools. Address Access Restrictions contains guidelines for dealing with supplier intellectual property while delivering open architecture models.

> *Example: The Statement of Work (SOW) for a program calls for the supplier to evaluate the openness of a mission system using a set of upgrade scenarios provided by the government. The Open Systems Management Plan from the supplier identifies three methods that will be used: Human inspection using a scenario-driven review reading technique, to be reviewed at PDR* [12]*; model-based virtual integration exercises for a subset of the MOSA scenarios selected by the initial reviewers, to be reviewed at CDR; and prototype update exercises during software and systems integration for a small subset of scenarios whose selection is guided by virtual integration risk assessments, to be reviewed at milestone C.*

2.2.3 Reduce Technical Risk

Technical risk management is needed for projects using new technologies or having challenging performance requirements. Key performance parameters that might not be satisfied are technical risks. Where a key performance value can be obtained by applying a suitable analysis to a suitable model, modeling & analysis can be used to support risk management. This requires identifying the uncertainties present in a model, determining how those uncertainties affect the analysis results for key performance values (called uncertainty propagation or sensitivity analysis [13]), and then reducing key uncertainties and identifying alternatives to mitigate unacceptable technical risks. Virtual integration exercises can increase assurance that technical risk mitigators will be effective and can be efficiently integrated or substituted if needed.

Key performance parameters having significant technical risk due to uncertainties in design parameters should be identified that can be determined by analysis of the architecture model. This analysis should be performed at the appropriate reviews, and risk analysis and mitigation should be done based on the analysis results. Later sections on Identify Variation Points, Configurations, and Dynamic Behaviors and Define Model Content Needed for Analyses include guidelines on modeling architectural alternatives, capturing uncertainty in models, and sensitivity analysis.

> *Example: Due to unusually stringent constraints on size, weight and power for a new Unmanned Air Vehicle (UAV), program planners have determined there is significant risk of software demand exceeding hardware capacity. The ACVIP plan says that the AADL model provided at PDR shall*

include demand and capacity estimates for the software and hardware components. The model shall be subjected to an analysis of weight, power, and hardware loading that shows sensitivity to the uncertain demand and capacity parameter values. The AADL model shall also include variation points that identify architectural alternatives having lower software demand (and the functionality and performance sacrificed for those alternatives) to mitigate this risk.

Example: A system integrator will be integrating several yet-to-be-implemented software applications from multiple suppliers onto a single processor module. Worst-case execution times that appear in the AADL component models at PDR were estimated using operation counts and benchmarks for similar components used on an earlier program. These model parameters have significant uncertainty. To mitigate the technical risk of overloading the processor when software is delivered and integrated, suppliers are directed to update their component models with improved estimates on a weekly basis between PDR and component delivery. The system integrator configures a continuous virtual integration server to pull component models from supplier repositories each week, virtually integrate them, and re-run the hardware capacity analysis. A dashboard display shows trends in software demands and hardware loading as the project proceeds after PDR.

2.2.4 Increase Affordable Capability

Reductions in project rework and duplicated work, improvements in technical risk management, and more agile and adaptive project planning and execution, can translate into either increased system capability within planned funding or cost savings for needed capabilities, depending on the acquisition objectives of the Program Office. Planners should consider ACVIP tasks that support identifying and managing these trade-offs.

Unplanned rework during software and systems integration is often dealt with by sacrificing planned capabilities in order to minimize cost and schedule overruns. ACVIP planners should consider categories of defects that have significant risk of being resolved during integration and acceptance testing by reducing or delaying operational capabilities, by work-arounds to operating procedures that increase crew workload, or by work-arounds to logistics and maintenance procedures that increase sustainment costs or reduce system availability [9].

An ACVIP architecture model of the system can be subjected to a variety of analyses in support of improved trade studies. Identify Variation Points, Configurations, and Dynamic Behaviors provides guidelines for modeling variation points useful for trade studies. Conventional trade space exploration automation methods can be applied to determine and trade-off various performance metrics. Advanced techniques such as continuous virtual integration with trade-off curve dashboards can be considered.

Agile, iterative, etc., development processes may improve combined, overall results (help push technology boundaries). Projects where requirements and desired capabilities may change during a project, or projects that have significant technical risk and require management of multiple technical risk mitigators, will particularly benefit.

2.2.5 Streamline Certifications

Failure to receive certification or readiness approvals is an important category of potentially expensive rework. ACVIP can be applied for the purposes of reducing defects that are not detected until late phase review by certification authorities identify shortcomings. If ACVIP outputs are also submitted as evidence to certification authorities, a much higher level of assurance is needed that the analysis results are correct and that the models accurately describe the as-built system.

In contrast to other goals, which can benefit from even modest improvements in defect detection effectiveness and risk management, certifications require a higher degree of assurance that models and their analyses capture the behavior of the as-built system. ACVIP plans for certifications should employ the baseline ACVIP architecture model to produce certification evidence, which reduces overall effort and increases assurance the evidence is consistent with the system built according to the model-based specification. Support Certification Approvals and Readiness Reviews provides additional guidelines where ACVIP plans include providing evidence in support of certifications and approvals.

ACVIP planners should consider categories of defects that have high risk of escaping into fielded systems with significant risk of consequential costs such as death, damage, or mission failure.

2.2.6 Benefit Future Upgrades and Families of Systems

Program Offices should consider life cycle management of ACVIP assets across the program as they develop and maintain their Program ACVIP Plans [6]. The ACVIP Acquisition Management Handbook provides guidance for overall program management [6]. This handbook provides engineering management guidelines to address these Program Office goals for both government and supplier personnel.

Anticipated upgrades to a system should be considered when developing both Program ACVIP Plans and ACVIP Management Plans. The intended uses of a delivered model should be considered, such as the analyses, configuration data generation, or model-based testing that model should support. Requirements on models delivered at the end of a contract should make those models useful assets for the next upgrade project. The Program ACVIP Plan should Identify Change and Configuration Management Procedures across the program life cycle.

Using delivered models, the government may supply future projects with a modified model that reflects government trade studies and is part of the specification of an upgrade to be performed. Planners should determine if a delivered model should also include variation points (architecture alternatives) in order to support future trade studies or risk management. Identify Variation Points, Configurations, and Dynamic Behaviors provides guidelines for modeling variation points.

A project may be upgrading a legacy system for which no models exist. ACVIP will require development of some models of the existing system in such cases. The cost of some model development may be recovered due to reductions in unplanned rework on a single project. There is a long-term benefit in developing and delivering models that benefit subsequent anticipated upgrades. In some cases, this may require more modeling and analysis effort than is necessary for a particular project, and the benefits

accrue over time. The Program Office and contractor should collaboratively Select Cost-Effective Modeling & Analysis. Guidelines for Mixed-Fidelity Modeling and Analysis are useful in these cases.

Program Offices may manage product lines or families-of-systems [14]. Consideration should be given to having a project contribute models or model updates to those government assets. The Program Office should consider requiring supplier trade studies tasks to support a determination of what modeling information belongs at the family-of-systems (product line) level versus at the deployed system level.

Consideration should be given to identifying and collecting metrics that will inform and improve overall effectiveness, efficiency, and risk estimation for future projects and other programs. Additional guidelines on this topic appear in Select Cost-Effective Modeling & Analysis.

2.3 Integrate with the Overall Development Process

ACVIP plans overlap with and should be consistent with many other plans. In addition to the System Engineering Management Plan, projects may require plans such as a Modular Open Systems Plan, Model Management Plan, Model Configuration Management Plan, etc. Define Model Content Needed for Analyses and Reviews provides guidelines to align ACVIP results with review milestones. Support Certification Approvals and Readiness Reviews provides guidelines to align with certification plans.

An Authoritative Source of Truth (ASoT) will include a variety of models written in a variety of languages for a variety of purposes. This handbook provides guidelines for using AADL models to perform ACVIP tasks. There will inevitably be information dependencies and requirements for consistency and traceability between AADL models and other kinds of documents and models. Identify Sources and Representations for ACVIP Data and Identify Relations Between Models provide guidelines for managing relationships between AADL and other kinds of models.

2.4 Identify Needed Skills and Training

The skills needed for ACVIP lie somewhere between those of traditional systems engineers and traditional software and hardware engineers. ACVIP requires some of both. ACVIP also requires model-based development skills to create, use, and manage models.

Systems engineers allocate stakeholder requirements, and other types of requirements such as regulatory and policy requirements, to system elements. They are responsible for mapping customer needs into an implementable and sustainable product. They perform trade studies, they define the system architecture and assure all its elements work together to meet stakeholder needs, and they identify uncertainties and manage risk. All these skills are needed for ACVIP.

Software and hardware engineers are familiar with the technologies needed to implement the system. Software and hardware integration skills are needed for ACVIP. Although initial ACVIP models may be fairly abstract, eventually key technical details must be modeled with sufficient precision to enable automated analysis.

Model-based engineering at the component level, such as models from which application code can be generated, is fairly mature and wide-spread. Effective model-based engineering of components requires

specialized skills and experience with the specific modeling languages and tools that are used. At the highest level of systems engineering where stakeholder needs are captured, models are still largely structured diagrams that are assured primarily by human review rather than by automated analysis.

ACVIP is a relatively new practice that falls between these two skill sets. ACVIP is a model-based bridge between stakeholder needs and an integrated system of implemented components that employs a relatively high degree of automation. Gaps in skills and experience should be identified and appropriate training performed prior to the start of ACVIP execution.

2.5 Select Cost-Effective Modeling & Analysis

Like software, modeling & analysis can consume an arbitrary amount of development resources if not properly scoped and managed. ACVIP planners should consider cost versus benefit given the available development resources and model analysis capabilities.

The primary way that ACVIP planners control costs versus benefits is through their selections and specifications of the analyses to be performed. It is recommended to do ongoing root cause analysis of defect databases to identify categories of defects that are to be targeted for early detection using ACVIP methods. The categories should be specified in a way that informs decisions about selecting and specifying the scope of analyses and the times at which they are to be initiated during development. Define Model Content Needed for Analyses and Reviews provides guidance on selecting and specifying different kinds of analyses.

Improving early defect detection is the primary ACVIP means to reduce rework cost, schedule and risk. Several studies have shown that detecting and fixing defects when they first occur during the initial phase can be between 30 and 1000 times cheaper than correcting them during integration testing, with corrections after initial fielding being even more expensive [15]. Studies indicate that even modest improvements in early defect detection can be cost-effective in reducing late phase rework [16, 17]. One study estimated that improving early defect detection by 10% would be cost-effective [18]. Some degree of false positive results is also acceptable. The goal is to achieve a good cost/benefit trade-off between early-phase effort spent and late-phase effort avoided.

Issue tracking and root cause analysis are common practice. Organizations typically do studies to categorize defects as a means to help improve development processes. When designing such studies, organizations should categorize defects in a way that informs ACVIP planning. Categorizations of defects should be developed based on the likelihood that available modeling & analysis methods and tools could detect those defects and the expected cost-to-repair for each category of ACVIP-avoidable defects.

> *Example: In a review of previous projects, a number of issues were found in the issue tracking system whose root causes were timing race conditions. Engineers estimated that only a third of these are likely to have been caught during CDR using available modeling methods and tools. ACVIP planners decide to perform the modeling & analysis that is feasible because (1) a large amount of time was spent in the system integration lab in previous projects to find the causes of intermittent timing issues, (2) repair required that multiple suppliers make changes to their delivered components, and (3) the costs of previous repairs exceeded the estimated modeling and*

analysis effort to be spent by a factor of ten, and engineers judged it very likely that more than 10% of such errors would have been detected using virtual integration methods.

Example: In a review of previous projects, a number of issues were found in the issue tracking system due to miss-matched variable and file import names between software source code units. The project plan calls for suppliers to deliver configurable source code, in a number of cases automatically generated using commercial tools. The software and systems integrator will configure delivered code for a specific system. ACVIP planners decide not to model Application Program Interfaces (APIs) in detail such as source code and file names because: (1) significant multi-organization collaboration would be required to create and maintain such detail in the models, (2) automatic application code generation limits supplier control over source code naming and data representation, and (3) the project plan makes it easy for the system integrator to make minor modifications such as name changes during software integration at little cost with little or no ripple effects.

A second method that ACVIP planners can use to control cost versus benefit is to vary the degree of detail and certainty to which modeling and analysis is performed. A uniform level of detail across the entire model is usually not necessary. Mixed-Fidelity Modeling and Analysis provides guidelines for modeling and analysis of different parts of the system model with different degrees of detail and uncertainty. Effort should be focused on the parts and aspects of the system where improved early defect detection has the greatest benefit.

A third method to control cost versus benefit is rolling ACVIP planning. An ACVIP Management Plan is a living document. The initial plan may call for itself to be updated at milestones. This is particularly useful when combined with plans for risk management. At each milestone, the risks due to uncertainty can be used to decide which portions of a model should be further detailed for which analyses at a subsequent milestone.

For first-time use of ACVIP methods, a program or organization may apply ACVIP as a shadow effort to train and gain experience with minimal impact on the baseline project while still providing some ACVIP benefits. However, planners should take into account that this compromises benefits. Previous experience with shadow applications is that ACVIP tasks often significantly lag baseline system development, compromising early defect detection and avoidance. Since a shadow model is not a build-to specification, removing defects in the shadow model does not always result in removing defects in the baseline system design and implementation. Metrics definition, collection, analysis, and evaluation must be done differently to account for these effects, including taking into account the learning curve for initial shadow application [19].

3. Structure Models for Delivery and Virtual Integration

A key concept of ACVIP is the delivery of models that are virtually integrated to form larger models. It must be possible to independently develop a set of models that can be delivered and integrated into a larger model. All these models must minimally satisfy the syntactic and legality rules of the AADL

standard. This requires appropriate structuring of the AADL system and component models. Descriptions for the models to be acquired must be developed first so they will virtually integrate into the system model. Model versioning and configuration and change management are needed. Define Model Content Needed for Analyses and Reviews provides additional guidelines to support various particular kinds of analyses for project review milestones.

3.1 Identify Sources and Representations for ACVIP Data

Different modeling languages and tools have been developed for different purposes. There are usually some fundamental concepts common to any given pair of languages, although the terminology usually differs. Some information can be captured in more than one language, but each language has unique strengths for that language's intended purpose. Different modeling languages have different tools available that do different things. Many modeling languages have alternative representations, such as DoDAF and AADL. The anticipated use case is that different models contain different but overlapping information that must be consistent with the ACVIP computer system architecture model. Tools are usually available to translate subsets of one language to subsets of another sufficient to support virtual integration of an overall embedded computer system architecture model.

Planners should identify guidelines for what information is appropriate to capture in what models and languages, how information flows and consistency is maintained between different models, and which model is the source-of-truth for which information.

Standard languages with standard semantics for a particular engineering domain should be preferred for that domain. MOSA goals may require delivery of models that the customer can use for analysis, trade studies, and to support upgrades to the delivered system. Such models should be in standard languages for which reasonably affordable and non-proprietary tools are available.

Assuring consistency between all the information captured in all models favors putting more information into fewer models. A single model used for multiple purposes should be preferred over multiple smaller models, each used for a single purpose, some used only once. This simplifies model lifecycle management. This provides assurance of consistency between the various analyses, tests, and other artifacts generated from models.

Virtual integration requires that models be structured so they can be independently developed and then virtually integrated to form a larger model. Different modeling languages are better or worse suited for this purpose. Languages that have a modular structure whose sub-models can be stored and exchanged independently are preferred. Planners should consider which modeling languages have been successfully used for virtual integration.

The Systems Modeling Language (SysML) is an Open Management Group (OMG) standard language developed for the domain of systems engineering [20]. Systems engineering is an interdisciplinary field that applies systems thinking and concepts; addresses operational, organizational, management, and lifecycle concerns for all stakeholders; and is general enough to apply to all engineering domains. A subset of SysML requirement, block, sequence, and state machine diagrams may be a source for high-level computer system architecture information.

DoD organizations are expected to conform to the DoD Architecture Framework (DoDAF) to the maximum extent possible [21]. DoDAF specifies information content and organization but not a specific modeling language. The Unified Profile for DoDAF and MODAF (UPDM) is an Open Management Group standard profile that refines the SysML profile with DoDAF concepts and is an accepted DoDAF representation. Some DoDAF system and operational views may be a source for high-level computer system architecture information.

Future Airborne Capability Environment (FACE) is an Open Group standard that includes a data modeling language to capture data architectures [22]. The language provides features to manage a standard shared and multiple domain-specific data models. FACE Unit of Portability (UoP) models can be a source of information about the data interfaces for software components. FACE integration models can identify data that is transferred between UoPs.

The Architecture Analysis and Design Language (AADL) is an SAE standard for modeling embedded computer system architectures. AADL provides standard semantics for integrated systems of software and hardware components. AADL supports use of a single computer system architecture model for multiple kinds of analysis, for software and systems integration, and for integrated system verification and testing. AADL supports automation of these tasks by applying multiple tools to an architecture model. AADL supports virtual integration of modeling information obtained from multiple other source models. This handbook uses AADL as the hub language for performing ACVIP tasks because it was developed and has been successfully used for this purpose.

The standard AADL textual format, which decomposes models into separate packages and files, is the recommended representation for model exchange. A host of tools originally developed for textual software source code can then be used for AADL, e.g., source code repositories, diff/merge, version and configuration management.

Individual software and hardware components are typically developed using a variety of modeling languages suited to each individual application and component. For ACVIP, the information needed from specialized component models describes aspects of their interfaces necessary for the planned virtual integration tasks. This is often obtained by scripts that extract that information in an AADL format.

> *Example: A supplier of a prototype mission system is provided with a Concept of Operations (CONOPS) document early in the Technology Maturation & Risk Reduction phase. The supplier is to deliver models that identify key subsystems and interfaces with Government Purpose Rights. They are to perform modeling and analysis and review the results at a combined SRR/SFR, PDR, CDR, and integration TRR. After establishing technical goals for each review, contractor ACVIP planners evaluated SysML, UML, AADL, and Modelica (all standard languages) against these requirements.*
>
> *The ACVIP Management Plan calls for SysML modeling of mission system capabilities and operational scenarios using the Unified Profile for DoDAF/MoDAF (UPDM). The SysML model will include system view diagrams that model the key interfaces and major subsystems and information flows of the mission system. Key interfaces and information flows are to be*

stereotyped using a SysML AADL profile. These portions of the SysML model are then automatically translated into AADL to establish requirements for various ACVIP models and results. The SysML model is the ASoT for the information that is captured in SysML. The generated AADL is to be incrementally extended and refined by embedded computing SMEs as additional detail is added for analysis and review at successive PDR, CDR, and TRR gateways. The AADL model is the ASoT for derived mission system architecture requirements, specifications, and analysis results that detect defects and verify requirements are satisfied.

These decisions were made because DoDAF and AADL are standards in their respective domains and satisfy MOSA requirements; AADL provides standard semantics and properties within the domain of embedded computer architecture with a range of analysis tools available; AADL is better suited for virtual integration due to the modular structure of the standard textual exchange format; and a single model can be progressively refined and subjected to multiple analysis tools as the project progresses through reviews and deliveries.

UML class and state machine diagrams were selected to capture detailed designs and generate code for software components that do fault management and message routing. Modelica models were selected to capture detailed designs and generate code for software components that do signal processing and feed-back control. Both of these are standardized languages that have semantics and available tools suited for the selected component application domains.

In previous contractor projects, different groups developed a large number of small spreadsheet models for different specialized purposes. This caused problems due to inconsistency between models and poor model lifecycle management. The ACVIP Management Plan states that the Project System Engineer must review and approve development and use of each spreadsheet model.

Example: The government is acquiring a software application for integration into three different types of air vehicle. To assure interoperability with existing software and meet MOSA requirements, the application shall conform to Future Airborne Capabilities Environment (FACE™) standards and a shared data model. There are also resource, timing, sequencing, and fault handling requirements that fall outside the scope of the FACE™ data modeling language standard.

ACVIP planners decide that a FACE™ Unit of Portability (UoP) data model will be developed by a software component supplier. A tool will be used to automatically translate the FACE™ UoP Supplied Model (USM) to an AADL component interface model. An AADL extension will be manually declared that uses standard AADL properties to add resource, timing and sequencing, and fault handling requirements. FACE integration models will not be used because engineers with prior experience said the effort required to fill gaps in AADL generated from FACE using extension and refinement is more complex than doing all virtual integration specifications in AADL. The FACE™ and AADL models will be provided as part of the specifications issued to the component supplier.

3.2 Identify Viewpoints to be Modeled

Architectures are described using one or more viewpoints [23]. Requirements, functional, data, software, and hardware architecture viewpoints are typically modeled, but not all in every modeling language used in a project. Many analyses can be applied to multiple viewpoints, but most cannot be applied to all viewpoints. Planners should consider which viewpoints should be subjected to which analyses and document that in the ACVIP Management Plan. That should guide decisions about what viewpoint content should be included in a model. All of the guidelines in this section apply to the different viewpoints.

AADL has limited standard **properties** to specify relationships between elements in different viewpoints. AADL binding declarations should be used only where the standard AADL semantics apply, such as software to hardware bindings. For other relations between elements in different viewpoints, the supplier should coordinate with the customer to arrive at a common AADL **property set** definition to be used. This **property set** should align with standard properties available in other modeling languages that are used to synchronize with AADL models.

> *Example: The System Engineering Management Plan calls for requirement and functional viewpoints to be fully modeled in SysML. Data, software, and hardware viewpoints are to be modeled in SysML at conceptual and logical levels-of-detail. The conceptual and logical data architecture model is to be translated to and synchronized with a model written in the FACE Universal Domain Description Language (UDDL), where it will be refined to platform level-of-detail and used as input to various analysis and code generation tools. In preparation for SRR, the SEMP and ACVIP Management Plans call for the SysML software and hardware and FACE data conceptual-level viewpoints to be translated to and synchronized with a model written in AADL, where they are virtually integrated to form a conceptual-level model of the integrated mission system. Traceability is established through all three models using an AADL **property set** based on the standard SysML traceability relationships. Static interface checking and resource loading analysis are initiated at SRR and incrementally re-executed thereafter. In preparation for PDR, the AADL model is resynchronized with the SysML and FACE logical-level models. Behavioral consistency analysis of communication protocols and component life cycles, Failure Modes and Effects Analysis (FMEA), and Risk Management Framework (RMF) step 4 analysis, are initiated and incrementally re-executed thereafter. In preparation for CDR, the AADL integrated system model is resynchronized with the latest SysML model and the FACE platform-level model and extended and refined to platform level-of-detail. Reliability Block Diagram (RBD) and schedulability analysis are added to the mix for review at CDR.*

3.3 Describe Models to be Developed and Delivered

A request for an item must describe the item to be delivered. The same is true for models. This section provides guidelines for describing a model that is to be developed or procured for an ACVIP task and purpose.

To describe a desired model, a model-based description should be used. This handbook uses the term “model-based description” rather than “model-based specification” to distinguish a simple model that helps describe a more elaborate deliverable model (model-based description) from a model that specifies

a deliverable system (model-based specification). A model-based description is used to procure a model; a model-based specification is used to procure a system. A Data Item Description (DID) or Contract Data Requirements List (CDRL) may have an accompanying model-based description, for example. The ACVIP plan for model-based descriptions resembles the supply chain structure: the government issues a model-based description to a system integrator, who develops and issues model-based descriptions to their suppliers, and so forth.

There is a high potential to reuse an early system specification model to describe a desired elaboration of that model as discussed in Abstraction, Elaboration, and Conformance. The terms "model-based description" and "model-based specification" may sometimes only distinguish an intended use of the same model.

In this handbook the singular "model" refers to any collection of AADL packages and property sets that satisfies the syntactic requirements of the AADL standard. The term "model" has compositional and elaborative semantics – multiple models may be integrated to form a system model, a system model may be decomposed into component models, and one model can be declared as an extension or refinement of another model. ACVIP plans must take the compositional and extensional relationships between models into account when identifying models to be developed and delivered.

> *Example: A development organization receives an AADL model from each of three suppliers at a review milestone. Each models a software component that will be integrated with other equipment to produce a system for an end customer. Two groups within the developing organization each create an AADL model for a piece of equipment developed internally. A third group then virtually integrates all these models to form a model of the system. This virtually integrated model is delivered to the customer for review. The ACVIP plans identify these as six AADL models (three models delivered by suppliers, two models of internally developed equipment, one virtually integrated model that includes the other five as subcomponent models). The plans call for each model to have no external semantic dependencies (other than standard AADL pre-declared property sets).*

Models should include comments or have associated documentation that explains the purpose, rationale, and intended use of the model – a user's guide for that model. Models should be readable. They should be structured and written to facilitate human review. For large and complex models, an overview of the structure of the model as a set of inter-dependent projects, packages, and systems should be provided. This may take the form of model configuration documentation.

ACVIP plans should identify analyses to be performed and the milestone(s) at which those analyses are to be reviewed. This is the primary method to specify what information content is required in a model – the model should contain the information necessary to perform the required analysis. Define Model Content Needed for Analyses provides guidelines for different kinds of analysis.

An important issue to consider is the reuse of existing models for existing components. Planners should evaluate the trade-off between modifying an existing component model (and component) to match the current system model (and system) versus modifying the system model (and system) to enable reuse of

existing component models (and components). ACVIP plans should reflect the overall project plans for the system and components themselves.

Three different patterns for model-based descriptions are described below.

3.3.1 Describe Models Using AADL Types

The simplest pattern to specify a desired model is an AADL **type** declaration provided as part of the description. The desired model is an AADL **implementation** that conforms to that type declaration. AADL **type** declarations may declare **features** (such as input and output message ports), **flows**, **properties**, operating **modes**, and **annex** declarations (such as error and functional behaviors) to which an AADL **implementation** for that type must conform. The **type** also unambiguously identifies the system boundary and its interface to its environment of use.

As a general guideline, customers should not override properties declared in **implementations**, and suppliers should not override properties declared in **types**. The keyword **constant** can be used to prevent inadvertent property overrides. Configuration parameters for configurable components can be treated as properties visible at the interface and declared in the **type** using non-constant **property association** declarations.

Example: A software component supplier is to provide a model that can be virtually integrated into an SRR model that will support a preliminary analysis of software memory loading and end-to-end latency requirements. The AADL type declarations shown in Figure 2 Type declaration included with a description of a desired implementation model *are developed by the system integrator and included with the description given to the supplier of this component model.*

```
data Sensor_Data
        -- Details omitted in this example
end Sensor_Data;

data Track_Data
        -- Details omitted in this example
end Track_Data;

system Desired_Component
-- The implementation model shall not refine any data types
-- of features or override any property values declared below.
        features
                sensed_objects: in data port Sensor_Data;
                fused_tracks: out data port Track_Data;
        flows
                sensed_to_track_latency: flow path sensed_objects -> fused_tracks
                        {Latency => 0ms .. 100ms;};
        properties
                Memory_Size => 5 MByte;
end Desired_Component;
```

Figure 2 Type declaration included with a description of a desired implementation model

This simple AADL **type** pattern may still be used in some cases where complex interactions with the environment are part of the requirements. Contracts, assume-guarantee, or input/output conformance techniques can encode certain behaviors of the environment as well as required responses from the desired system. AADL Behavior Annex declarations may be used where the ACVIP Management Plan calls out appropriate conformance guidelines and usage [24].

3.3.2 Describe Models Using AADL Environment Models

A second pattern is to explicitly model elements of the environment in which the desired system will be used. In this pattern, the AADL **system** used to model the system being developed is an AADL **subcomponent** within a larger AADL **system** declaration. Sometimes this outer environment model represents a physical environment, and elements such as crew and external objects are represented as subcomponents. AADL **abstract** components should be used for objects that fall outside the scope of AADL semantics, such as crew members.

> *Example: Figure 3 Environment model included with specification of model to be delivered illustrates a model that includes a **system subcomponent** for the system being developed (the mission_system) together with **abstract subcomponents** that represent crew, an external network, and terrain that are not part of the system but are necessary to specify interactions that the system must support. The overall system environment is declared using the **abstract** rather than the **system category**.*

An environment description model should be usable as a virtual test harness by suppliers of component models. A supplier of a component model should be able to perform a local virtual integration of their component model into the provided environment model. The environment description model should declare **flows**, **properties**, etc. that identify the analyses that should be supported by the delivered component model. Many analyses depend on information from the environment-of-use of a subsystem or component. Detailed guidelines for different analyses are found in Define Model Content Needed for Analyses. It should be possible for a supplier to run required analysis tools on this local virtual integration, even though the results will not be as complete or certain as those to be obtained by the model integrator during the system model virtual integration task.

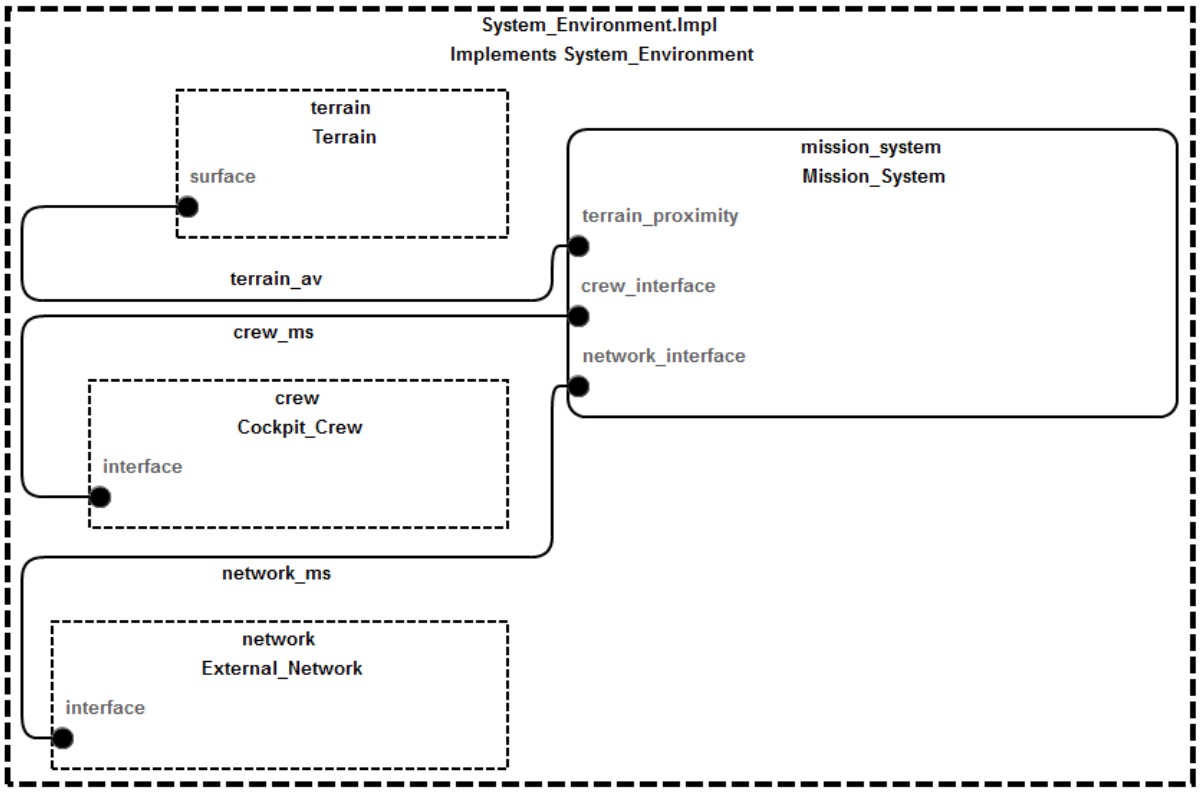

Figure 3 Environment model included with specification of model to be delivered

3.3.3 Describe Models Using a Template

A third pattern for model-based descriptions is a pattern or template model to be elaborated to create the desired model. Both of the above patterns (type and environment) may be viewed as starting points for model elaboration. More complicated scenarios are also possible, such as declaring a partial AADL **implementation** that is to be **extended** and **refined** by the to-be-delivered model. Guidelines for elaborating models may be found in Abstraction, Elaboration, and Conformance.

AADL allows a **property** declaration in one part of a model to override **properties** and other characteristics of model elements that they reference or incorporate. However, in some cases declarations in a model are specifications that should not be overridden in supplier models. Some properties in supplier models are not configurable by (should not be changed by) the virtual integrator. It is recommended that the AADL **constant** keyword be used in property value associations to explicitly restrict re-definitions of such property values. **Property associations** declared in **types** that specify requirements to suppliers that should not be modified by suppliers should be declared **constant**. **Property associations** declared in **implementations** that are fixed design choices made by suppliers and not configurable by users should be declared **constant**. ACVIP plans should describe what may be overridden when model elements developed in one task depend on model elements developed in another task.

> *Example: The **type** declaration in Figure 2 Type declaration included with a description of a desired implementation model includes **constant** property associations and a comment that the supplier model shall not override any **property** values or **refine** the **classifiers** of any **features** declared in*

*the AADL component **type** declaration – they are specifications to which the supplier must conform.*

3.3.4 Provide Common AADL Libraries

Property sets and libraries of common elements to be used by multiple model suppliers may also be useful to include with descriptions of models to be delivered. This is particularly true where the system is to conform to specified standards. Where a system is required to conform to a standard at a key interface, the model of that system should capture that requirement.

*Example: The AADL ARINC 653 Annex defines a standard way to model ARINC 653 compute modules. This annex defines a standard AADL **property set** for ARINC 653 standard properties. This annex provides a modeling pattern to be used for the architecture of an integrated ARINC 653 compute module. Tools developed by different vendors to support ARINC 653 modeling will all recognize the same properties and patterns. ARINC 653 process and partition models developed by different suppliers will all virtually integrate into an ARINC 653 compute module model.*

*Example: A Mission System Integrator (MSI) is procuring software applications from several subcontractors. These applications are to be hosted on a common computing platform provided by the MSI. The MSI develops an AADL **package** that declares a platform execution environment model at a sufficient level of fidelity that software suppliers can bind their models and run analyses to do static checking of interface consistency and resource loading. This package is provided to suppliers as part of the descriptions for models they are to deliver for PDR.*

3.4 Modularize Model Text and Diagrams

Models themselves are modularized into declarations, files, folders, etc. This is not the same thing as the modularization of the system into an assembly of components – that is captured in a model, but a model also has its own structure as a set of text strings and diagrams and files. The modularization of a system is described by an **instantiation** of a selected AADL **system implementation** declaration. ACVIP plans should address how the model text and diagrams are modularized for separate development and delivery and virtual integration.

AADL textual representation should be used for model delivery. The textual grammar is what is standardized. The AADL textual grammar and structure allow many methods and tools used for software source code change and configuration management to be applied to AADL models. Methods and tools for source code delivery and sharing can also be adapted for AADL text. The current AADL convention is that different AADL Integrated Development Environments (IDEs) may provide different graphical viewpoints (different types of diagrams) obtained by round-tripping from the AADL textual representation.

AADL text may be modularized using the following language and development environment features.

- AADL **packages** (including **property sets**) are collections of **type**, **implementation**, and **property definition** declarations. The language definition requires all **type**, **implementation**, and **property** declarations to appear within a named AADL **package**. **Packages** may depend on other named

packages. Declarations in one **package** can **extend**, **refine**, and add **properties** to **type**, **implementation**, and **property** declarations found in other **packages**. A **package** or **property set** should be the smallest unit of model development for the smallest unit of development task break-down.

- Development environment files are the smallest unit that can be stored in a repository, exchanged between developers, and managed in a change and configuration management system. Each AADL **package** should be stored in its own file.
- Development environment projects are sets of related inter-dependent **packages**, **property sets**, files and folders. Many AADL tools, change and configuration management tools, and delivery tools and procedures, define and support a project concept, although with varying capabilities and terminology. Development environments should allow a project to have information dependencies on other projects. A project is the recommended unit for model delivery.
- AADL **system implementation instantiations** are representations of the structure and behavior of a specific system. A **system instantiation** can be automatically generated into a file from a selected **system implementation** declaration found in a collection of **packages** and **property sets** that have no unsatisfied external dependencies. A **system instantiation** file is a common unit that is input to a tool for analysis. System models that are to be subjected to individual analysis should have an identified AADL **system implementation** declaration that can be instantiated.
- Different change and configuration management methods and tools may have differing conceptual models and capabilities (e.g., distributed repositories, change sets). Where the involved parties use different methods and tools, the ACVIP Management Plan should describe how versioning and configuration information is to be exchanged along with the models.

ACVIP plans should establish naming conventions for deliverable units (e.g., packages, files, folders) as needed. These are only needed for elements of the model that will be referenced from other models during virtual integration or for change and configuration management purposes.

The recommended default unit of model development and delivery is a set of AADL packages organized as a set of one or more development environment projects. Individual AADL **packages** may be appropriate for some purposes.

Different development environments may add files to projects or assume additional usage conventions. This applies to AADL Integrated Development Environments (IDEs) used for model development, versioning conventions used by different organizations, and change and configuration management tools. AADL projects should be structured to be as robust as possible to such differences. ACVIP plans should identify differences that do have an impact and how they are addressed.

When defining model development and delivery units, ACVIP planners should take into consideration the following issues.

- Performers responsible for capturing specific data in models
- Temporal sequencing of data availability and model development and use
- Project plan dependencies between tasks and their input and output models

- Analyses that require a **system instance** to be generated
- Information dependencies and relationships between models
- Intellectual property and information security boundaries and restrictions
- Model delivery and sharing milestones and procedures and methods
- Model versioning conventions and change and configuration management methods
- Alignment with system and component versioning and change and configuration management

3.5 Address Access Restrictions

Information access restrictions must be considered when modularizing model content for delivery. Sufficient access rights must be provided to the model integrator by all component model suppliers to perform the planned virtual integration and analysis tasks.

Architectural models should primarily focus on interfaces, externally observable behaviors, and interactions between components. Information about component internal designs should be minimized. Among other benefits, this simplifies dealing with access restrictions, which are more likely to apply at the detailed design rather than the architectural level-of-abstraction.

Because a **package** is the smallest unit of model exchange, **public** and **private** sections within the same package should not be used to satisfy data access restrictions. When a model integrator is developing model descriptions for model suppliers, a goal is to reuse elements of the system model to produce model descriptions for model suppliers as discussed in Describe Models to be Developed and Delivered. It may be necessary to derive different sanitized model descriptions for different model suppliers due to information access restrictions. Where this is necessary, the structure and modularization of the system model should take this into consideration so that deriving and managing sanitized model descriptions is easier. Identify Change and Configuration Management Procedures should include these derived component model descriptions.

A model integrator may have permission to access data from component model suppliers but be prohibited from sharing component models from one supplier with another supplier. The model integrator may not allow portions of the integrated model to be seen by component model suppliers. The overall set of models should be modularized so there is little or no direct dependence between models from different suppliers. Note this situation complicates collaborative debugging of the virtually integrated models as discussed in Plan Virtual Integrations.

A component model provider may be required to support certain analyses of the virtually integrated system model but not want to share certain details required for that analysis. A method that can accomplish this for certain kinds of analysis is for the component model provider to develop two models, an internal fully detailed model and a sanitized component model delivered for virtual integration. The component model developer performs an analysis on the fully detailed model and then annotates the sanitized component model with component analysis results that are needed to run that analysis on the overall virtually integrated system model. See Abstraction, Elaboration, and Conformance for a discussion of abstraction relations between models. This approach requires an analysis tool that supports this form of compositional or gray-box analysis.

Example: A Mission System Integrator (MSI) will conduct agile continuous virtual integration process with multiple suppliers. The MSI directs each supplier to establish a model repository that can be used to securely exchange models between MSI and supplier. The MSI will Describe Models Using a Template plus other documentation for each supplier by downloading that material to each supplier's repository. Each supplier sees only the template-based description of the model they are to deliver. Using modifications of tools commonly used for continuous software integration testing, the MSI stands up a continuous virtual integration server that automatically pulls models from each supplier's repository, virtually integrates them into the MSI's overall architecture model, and applies a selected set of analysis tools. Only the MSI has visibility to all supplier models and the overall system architecture model. The MSI and their suppliers use a collaborative agile process in which all parties continuously update their models. The MSI configures the virtual integration server so that dashboard displays and error notices appropriate for each supplier are visible to that supplier.

3.6 Identify Relations Between Models

ACVIP uses many models and work products that have a variety of relationships to each other. ACVIP plans should identify important relationships between the multiple AADL models and other work products and how those relationships are captured, managed, and verified. This subsection presents considerations and guidelines for four important classes of relations between models.

Many sections of this handbook cite the use of AADL **extends** and **refines** declarations, and **property** inheritance and override language features, to manage a variety of relationships in large, complex, evolving virtual integrations of multiple models from multiple sources. Such models typically consist of many projects and **packages**. Selection of an AADL Integrated Development Environment (IDE) and user training should take into consideration the cross-referencing capabilities for these language features.

3.6.1 Dependence

In this handbook the singular "model" refers to any group of AADL **packages** and **property sets** that satisfies the syntactic requirements of the AADL standard. A model that is syntactically correct may still have semantic dependencies on other models. The set of models input to an ACVIP task must typically be syntactically correct and semantically self-contained -- that set must collectively satisfy all the standard AADL legality rules, and it must be possible to instantiate system implementations declared in that set.

A dependency of AADL **packages** or **property set** on another must be explicitly declared at the beginning using a **with** declaration. Almost all AADL tools will require that **package** dependencies be satisfied and will perform legality checks across **package** and **property set** boundaries. A project is a set of AADL packages. A dependency exists between two projects if one project contains a package that **with**'s one or more **packages** or **property sets** in the other project. An AADL model is a set of one or more AADL projects. A model is dependent on another AADL model if it references things declared in that other model.

A model developed by one organization may depend on models developed by other organizations. ACVIP Plans should identify these dependencies and explain how they are to be satisfied.

The AADL language definition allows circular dependencies between **packages**, but not all development environments may support circular dependencies between projects. Circular dependencies between projects should be avoided unless there is a special need and all participants' tools support this.

> *Example: A program plans to use a shared data model to specify message content and layout across multiple suppliers. ACVIP plans state that all AADL models developed before the delivery of that shared data model should not explicitly identify an AADL **data type** in declarations of component input or output message ports (AADL permits this sort of partial declaration). AADL **extension** and **refinement** declarations should be used to add this information later when the data model **package** becomes available. This plan satisfies dependencies by limiting declarations so they do not create unsatisfiable dependencies until that information becomes available.*

> *Example: A program plan calls for a supplier to develop a component model that interfaces with other component models to be provided by other suppliers. ACVIP plans state that the customer shall provide as part of the model-based description to each supplier a mock model of the interfaces to other supplier's components sufficient to pass the semantic dependency rules of AADL. This plan satisfies dependencies by providing mock models that are sufficient for component model development.*

The type compatibility rules of AADL distinguish between the same **type** and an **extension** of that **type**, a distinction that may be meaningful for type-checking tools. A **renames** declaration is needed to refer to the same **type** across **package** boundaries. Where an AADL **type** declaration appears in a different **package** from its AADL **implementation** declaration(s), a reference to an external AADL **type** should be made using an AADL **renames** declaration rather than by declaring a local AADL **extension** of the external **type** unless there is a specific reason to introduce a new **extension** of that **type**.

3.6.2 Abstraction, Elaboration, and Conformance

When one model is used as a specification for a system, and there is a more detailed model that does or will exist for that same system, then the more detailed model should be consistent with or satisfy the less detailed model. One case where this will occur is successive virtual integrations at SRR, SFR, PDR, and CDR. The model used at SRR will be less detailed than the one used at PDR, which in turn will be less detailed than the one used at CDR. All three of these are models of the same to-be system. They just have different degrees of information content and uncertainty. Another case this will occur is when an abstract model is included as part of the specification for a more detailed model to be acquired from a supplier, as discussed in Describe Models to be Developed and Delivered. In many cases the more abstract model is intended to capture requirements that must be satisfied by the more detailed one.

In this handbook, the less detailed and more uncertain model will be called an abstraction of the more detailed and certain one. The more detailed and certain model will be called an elaboration of the less detailed and more uncertain one. Intuitively, when model E is an elaboration of model A, then E should satisfy or be consistent with or conform to A in some sense.

An abstraction is a model *A* that has some, but not all, of the information in another model *E* and where a class of properties that are true of *A* are also true of *E*. Sometimes *A* is developed first using limited available information and is subsequently changed into *E* as additional information becomes available. Sometimes the more detailed model *E* is developed first and then *A* is developed because *A* enables more tractable analysis of a certain class of properties.

The key idea in the preceding paragraph is that *E* conforms to *A* when properties of interest that are true for *A* are also true for *E* – *E* satisfies every requirement that *A* does. ACVIP uses virtual integration analysis to provide assurance that the desired properties for *A* still are true for *E* as the model is elaborated during development. An ACVIP Plan should identify analyses that are to be performed across all reviews, with increasingly detailed and certain analysis results, as one way to define what it means for successive models to conform with preceding models.

A second way to declare conformance requirements is to use AADL **extends** and **refines** declarations.

Formal conformance relations are recommended where feasible and reasonable, as their rigor avoids ambiguity and can enable automated verification [25]. Where used, ACVIP plans should identify how a conformance relation is (to be) defined. The plan should identify how compliance with that relation is to be verified. There may be multiple conformance relations required between a pair of models.

> *Example: An AADL CDR model includes an abstract state machine specifying how a **type** of component responds to different kinds of arriving messages. This is specified using language features from the standard AADL Behavior Annex* [26]. *ACVIP plans state that all component **implementation** models developed by suppliers and all models of environments into which they are virtually integrated shall satisfy an input/output conformance relation[1], where the formal definition of the input/output (I/O) conformance relation shall be provided with the AADL model* [24]. *ACVIP plans state that a sample of components shall be tested before delivery to assure I/O conformance of the as-built components with the AADL **type** and behavior specification, where the test set shall achieve a given model-based test coverage metric.*

A model is also an abstraction of the system that it describes. This meaning of conformance is discussed in Assure System Conforms to Models.

3.6.3 Layering, Extension, and Refinement

AADL has language features that support two important kinds of relations between models: extension with refinement to support elaboration of earlier models by adding more information; and bindings to support layered architecture models. An advantage of using these language features is that syntactic and

[1] I/O conformance relations can be applied when a specification model and an implementation model take the form of state machines or transition systems whose events are classified as either input or output. The specification can be viewed as a game played between the component and its environment. An input move by the environment may change the state of the component, the component may only respond with an event allowed in its current state, and some of the allowed output moves may also change the component state.

legality rules of the language provide some assurance of conformance between different layers or degrees of abstraction as an overall system model increases in scope and detail.

Figure 4 Extension with Refinement Adds Information to a Type (or Implementation) illustrates how extension with refinement can add new information to existing models in a way that is consistent with AADL legality rules. The **system** NavLogical is an AADL type (interface) declaration that is subsequently extended to **system** NavPlatform by refining the features to declare details about groups of messages sent and received and to change property values declared in the parent type. The AADL standard specifies various forms of refinements that can be applied to different kinds of feature and subcomponent declarations when they are refined in an extension.

This figure illustrates two ways in which property associations can be declared: in a **properties** section of a type or implementation declaration (the example here is the `Memory_Size` property association); or as a clause in a feature or subcomponent declaration (the example here is the `Data_Size` property association). AADL rules for overriding property values give higher precedence to property associations on feature and subcomponent declarations than to property associations in a type or implementation **properties** section. If a subcomponent declaration has an inherited property association and is refined to have an implementation with a property association, the inherited property association will override the one in the implementation. To avoid possible confusion, either one pattern or the other (declare property associations on features and subcomponents; or declare them in **properties** sections) should consistently be followed for a given property. Property associations in property sections of types and implementations are more flexible as models become larger.

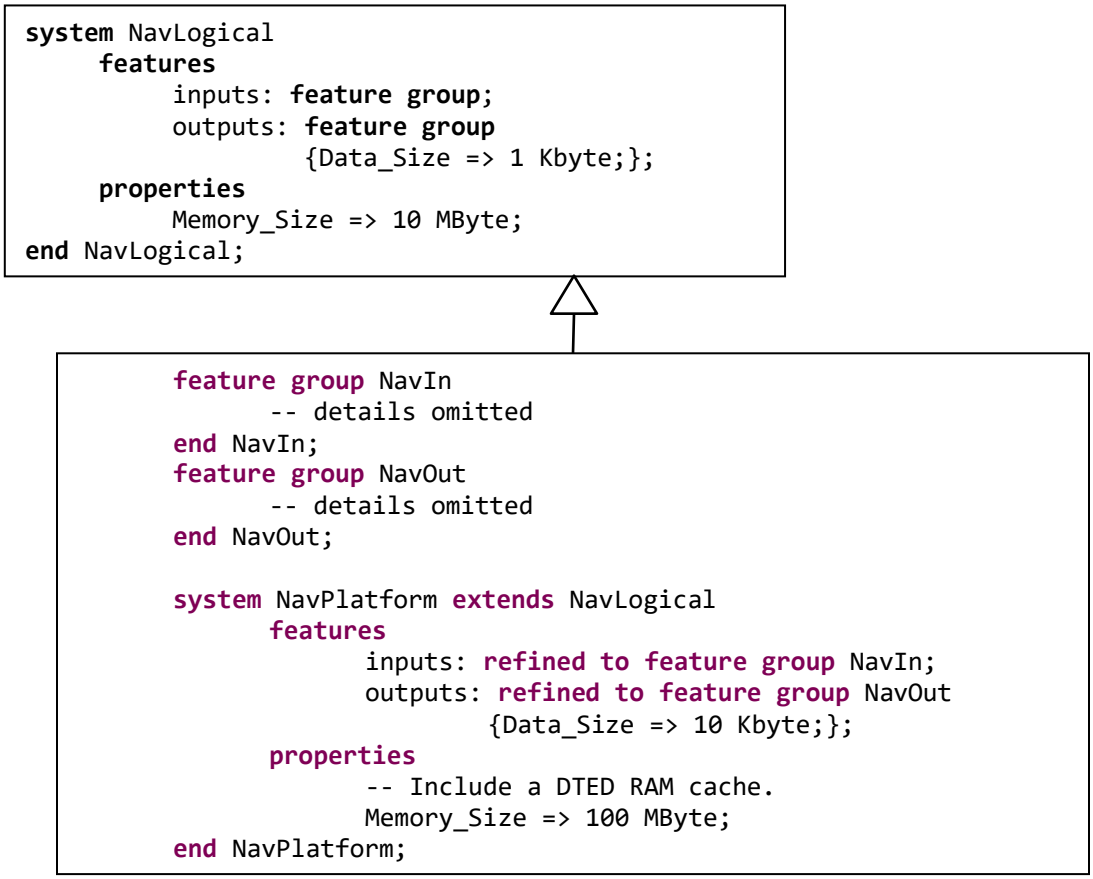

Figure 4 Extension with Refinement Adds Information to a Type (or Implementation)

Figure 5 Layering is a Common Pattern in Architectures and their Models illustrates how binding properties can be used to model layered architectures. This example illustrates the use of the `Function_Binding` property to allocate abstract and system types used to denote functions to process and data software components that provide those functions. The `Processor_Binding`, `Connection_Binding`, and `Memory_Binding` properties can subsequently be used to bind these software components to virtual resources in an execution environment that are themselves bound to physical hardware elements. The layers in this figure illustrate the concept; the layers in a model should be selected based on the needs of the individual project.

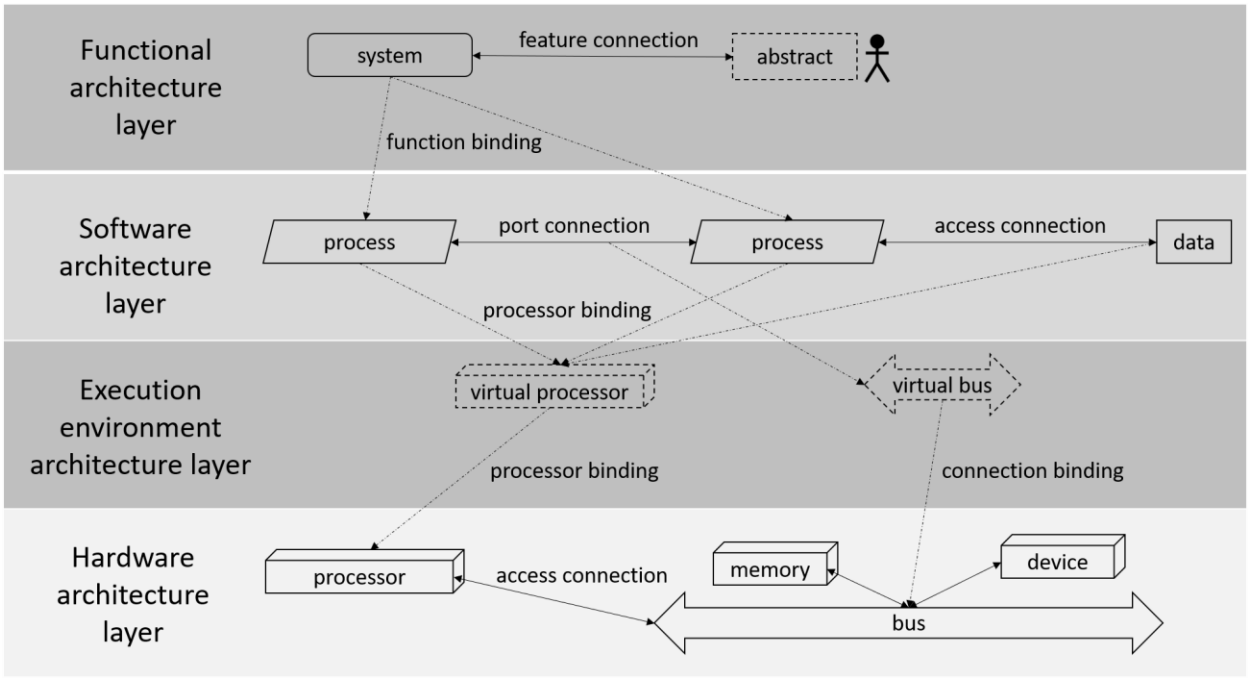

Figure 5 Layering is a Common Pattern in Architectures and their Models

Consideration should be given to structuring layered models so that different layers can be developed independently and then virtually integrated by adding an appropriate set of binding properties from one layer to another. This is another useful pattern when describing models to be procured from different groups for subsequent virtual integration.

Figure 6 Implementation, Extension and Refinement for Layered Architectures illustrates how these language features can also be used when a layered architecture model is developed. Instead of binding elements declared in one layer to elements in another layer, an element in one layer can be implemented using subcomponents that are considered to be in another layer. Extensions and refinements may also be assigned to different layers than the layer in which the parent elements are assigned. The choice depends on the nature of the project. For example, if there is a relatively straightforward mapping of functions to software components, then implementing extensions and refinements of the original elements used to model functions is likely the simpler approach.

Decisions about the modularization boundaries at which extension and refinement are used should be made carefully. Excessive use of these features can make a model unnecessarily complex to understand and review because the information that applies at the more detailed levels of modeling is spread across one or more parent declarations. Consideration should also be given to the restrictions on changes that can be made using refinement declarations due to AADL legality rules. For example, occasionally it may be necessary, due to restrictions on refinement, to render an inherited flow implementation ineffective by reassigning properties to benign values and then declare a new one.

It is recommended that functional decomposition eventually be performed to a level-of-detail where individual lowest-level functions are allocated to a single component, so that the implementation or realization of a lowest-level function is not split across multiple components.

When layered architecture models are used, and portions that are to be virtually integrated appear in different layers, ACVIP planners should be careful to describe the architecture layers, the layer to which different procured models are assigned, and the language features used to interface between different layers. Models should be structured for delivery so that binding and extension declarations are made in a separate package by a different group as needed. Note that layered models may be incrementally delivered. That is, an ACVIP Plan may call for initial delivery of layers with subsequent delivery of more detailed models of the layer.

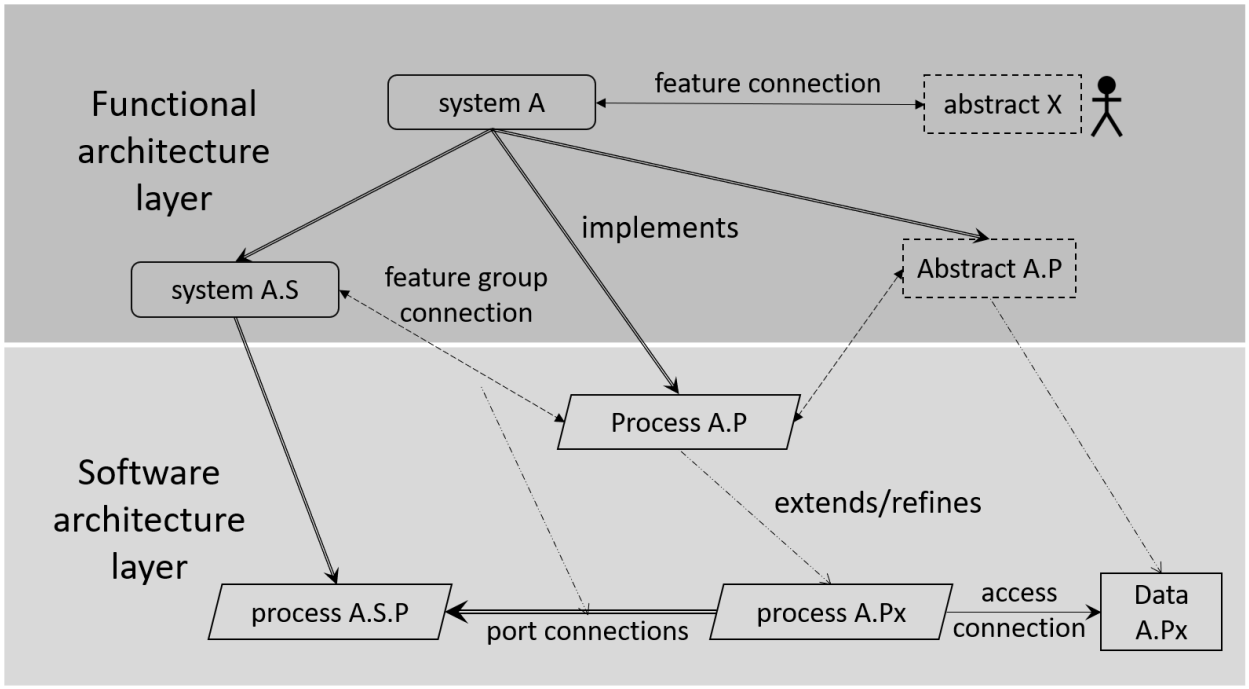

Figure 6 Implementation, Extension and Refinement for Layered Architectures

3.6.4 Conceptual, Logical, and Platform

A number of standards use the terms conceptual, logical, and platform to refer to different levels of abstraction or detail in a model. The AADL standard does not. Although alignment with AADL categories may be reasonably intuitive in some cases (e.g., **abstract** components and abstract **feature** features naturally align with a conceptual level-of-abstraction), in other cases this may vary depending on the intended model use and detail (e.g., **connection** may align with any of conceptual, logical, or platform depending on context and details such as the connected features, data types, and bindings). Planners should determine if alignment with conceptual, logical, and platform terminology is needed for a project and, if so, provide guidelines for this alignment.

3.6.5 Sources of Truth

A model may contain data that is redundant with or derived from data in other models. ACVIP plans should identify which models are considered the "single source of truth" for such data. Ideally other models will simply reference the single source of truth for a particular datum. There may be a more complex relationship between the single source of truth model and other models that include somewhat redundant information. Any of the relationships discussed in the following paragraphs may be used for this purpose.

> *Example: A model developed for SRR declares mass budgets for components of a system. ACVIP plans state that any mass data in subsequent more detailed models must fall within these budgets. The plans state that the "single source of truth" for mass properties data used for CDR analysis shall be the 3D Computer Aided Design (3D CAD) solid models. Any mass properties appearing in the AADL CDR model for a hardware component should match the values obtained by performing a mass properties analysis of the corresponding version of the component solid model. The plans state that model consistency analysis shall be performed to verify that specific mass properties in all AADL models fall within the budgets declared in the SRR model and that these values are equivalent (within allowed error bounds) to the mass properties obtained from the solid model.*

Information may be captured in multiple models, especially models in different languages. For important pieces of information, the plan should identify which model is the single source of truth with which other models should be verified to conform. Plans may need to take into account differences in how a piece of information is declared in different models. For example, an estimated value declared as a scalar in one model may need to conform to a budget declared as a range in another model.

Complex relationships may exist between declarations within a single system model that resulted from virtual integration of multiple components. Describe Models Using AADL Types discusses the need to control which properties can be overridden and which not by different roles. More complex relationships may exist due to Abstraction, Elaboration, and Conformance and Mixed-Fidelity Modeling and Analysis. For example, a property in a model description may establish a range or budget for properties to be declared in the model supplied to satisfy that description. Care should be taken to clarify any such detailed relationships between properties in different models or parts of a model.

Consistency is used in this handbook to refer to any rigorously defined relationship between models that has been identified in ACVIP plans but does not fall into the previous categories of model-to-model relationships. The ACVIP plan should cite definitions for any additional consistency relationships and how those relationships are used to carry out ACVIP tasks.

> *Example: An AADL PDR model declares a logical structure for the hardware of an embedded computer system using AADL concepts such as **processors**, **buses** and **memories**. A 3D CAD solid model of the air vehicle includes parts, assemblies, and mating constraints for circuit cards, chassis, wiring harnesses, and electrical connections. The logical resources and **connections** in the AADL model must be consistent with the assemblies and electrical connectivity in the solid model.*

*Example: A Simulink model declares sampling rates for blocks in a control algorithm specification. Code generated from this model is modularized for execution by different periodic **threads** that are dispatched at the specified sampling rates. The set of **threads** declared in an AADL CDR model of this software component must be consistent with the Simulink sampling rates and control code modularization. The plan states that the Simulink model is the "single source of truth" for sampling rates.*

3.7 Identify Variation Points, Configurations, and Dynamic Behaviors

There are several situations where different kinds of configurations and behaviors may need to be captured in a single AADL model.

- A model may declare multiple possible configurations in order to describe a family of related systems across multiple acquisitions or to declare alternatives for trade-studies. The model is configured by choosing variants for variation points identified in the model.
- Components delivered by suppliers may need to be configured by the system manufacturer during system integration or during field maintenance. The model declares how components are to be configured (what configuration data needs to be applied to each component).
- A delivered system may undergo run-time architectural reconfiguration during mission planning and preparation or mission execution.
- A delivered system and its components exhibit a variety of behaviors during use. Users may choose between different system functional modes at run-time.

This handbook distinguishes these situations, and this section provides modeling and analysis guidelines for each.

3.7.1 Configurable Models and Variation Points

A configurable model contains variation points, places where a declaration can be modified in a defined number of ways. Given a choice for every variation point in a model, each selected from a declared set of alternatives for each variation point, a single modeled system instance can be generated for a specific system configuration. This represents a development-time configuration to select among alternative systems, not a possible run-time reconfiguration. The final system conforms to a single selected model configuration. A configurable model can be viewed as a function that maps a set of choices for a set of variation points to a specific AADL **system instance** model. Configurable models support product line management and trade space exploration, for example.

Suitably designed trade studies can be a useful way to assess the degree to which MOSA objectives are met. Upgrade scenarios can be used to guide the identification of variation points. The identification and creation of model variations and virtual integration exercises may be as informative as the analyses performed, as those are virtual system modification exercises.

It is rarely the case that all possible combinations of variation point choices result in an acceptable system configuration. The set of analyses that ACVIP planners decide to perform on a model implicitly constrains which combinations of choices are allowed (pass all analyses). A configurable model may also explicitly define constraints involving multiple variation points, which also has the effect of ruling out alternatives that appear in the cross-product of all possible variation point choices.

There is no standard way in AADL to explicitly identify which declarations are intended to be developer-selectable variation points (and which are not) and declare allowed sets of alternatives for variation points. This should be documented, e.g., by defining specific modeling conventions in the plans, in comments within the models. A number of standard AADL declarations may be identified as variation points. Some primary candidates are:

- **Property value associations** with an identified set of alternative values
- **Subcomponent** declarations with an identified set of alternative **types** or **implementations**
- **Subcomponent** declarations with identified sets of alternative actuals for **prototype parameters**
- **Array** declarations with an identified set of alternative **index ranges**
- Multiple **system implementations** that could be **instantiated**

Variation points and their associated sets of alternative choices should be explicitly identified in the model. There may be configuration choices that cannot be easily defined using the above methods, such as alternative patterns in an identified set of **connections**. This could be done using defined comment formats. Macro and language extension methods and tools could be applied. In all cases, it should be possible to unambiguously preprocess such models by an appropriate method or tool to generate a legal AADL model for a selected configuration. This generated model will typically have a single **system implementation** declaration used to create an **instance** model for that system configuration.

AADL **mode** declarations should not be used to specify development-time configurable models. They should be reserved to specify possible dynamic re-configurations of fielded systems as defined in Configurable Systems. AADL Behavior Annex declarations are similarly reserved to specify run-time behaviors as described in Alternative Functional Behaviors.

> *Example: Developers are trying to decide between an architecture that uses three large enclosures with point-to-point very-high-speed connections, versus an architecture that distributes a larger number of smaller enclosures throughout the vehicle and uses a switched network. An AADL PDR model is created that contains declarations for both kinds of computing platforms.*
>
> *To explore the trade space, developers use a trade space exploration framework that integrates several tools. One tool interprets **property associations** and certain forms of **subcomponent classifier** declarations in an AADL model as variation point declarations, where the **property type** and the set of available **implementations** for a **classifier** define sets of alternative choices. A trade space exploration tool uses Monte-Carlo methods to generate combinations of choices. For each choice, a tool is applied to the configurable AADL model to produce a **system implementation instance** model for that set of choices. Weight analysis, power analysis, utilization analysis, reliability block diagram analysis, and fault tree analysis tools are automatically applied. A trade space visualization tool inputs analysis results for all configurations and provides interactive visualizations of the trade space Pareto frontier to the developers.*

3.7.2 Configurable Components

Some components require the system integrator to provide configuration data for that equipment. Field maintenance may sometimes reload configuration data. Configuration of each individual piece of equipment may be performed in a number of ways, such as switch settings or installing a configuration file.

The use of standard AADL properties to specify component configuration data should be very carefully assessed to ensure the purpose is consistent with standard AADL semantics for those properties. Otherwise new **properties** should be defined for that **type** of component in an AADL **property set**. Where the configuration data is complex, a **property** can name a file that contains the configuration data. Where complex configuration data includes information needed for a desired architectural analysis, a user-defined AADL **annex** can be developed for that **type** of component. The model should be documented to identify which **property** or **annex** declarations specify configuration data to be applied to components during manufacture.

ACVIP plans should describe how alternative component configurations are to be handled during virtual integration analysis.

3.7.3 Configurable Systems

AADL **operating modes** should be used to specify how a system may undergo architectural reconfigurations during use – changes during operation to the set of **subcomponents** or **connections** or **properties** that may affect architectural qualities such as timing or safety or security. These may occur during maintenance, mission planning and preparation, or during mission execution. An AADL operating mode should not be confused with a functional mode, to be discussed in Alternative Functional Behaviors.

In AADL **type** and **implementation** declarations, AADL **modes** may be declared together with **events** that cause transitions between **modes** during system use. The collection of all AADL **mode** declarations in all components together with **event connections** between components forms a concurrent state machine model, where transitions between operating **modes** occur when specified **events** occur. Many AADL declarations have a **modes** clause to specify whether they apply or how they apply when a component is operating in a declared subset of its **modes**. AADL **modes** should be used to specify how the architecture-level behavior of a system may change after that system has been fielded. These alternative behaviors are called system **operational modes** rather than model configurations.

Because AADL **modes** declare changes to operational behavior, **mode** declarations affect many architecture analyses. Because the set of all **mode** declarations forms a concurrent state machine model, state space explosion (or in this case **mode** space explosion) can easily result in intractable analysis for complex patterns of potential **operational modes**. **Mode** transitions often have transient semantics that need to be taken into account during analysis, which complicates and may limit the results of certain kinds of analysis.

> *Example: A **mode** transition that activates and deactivates sets of **threads** is not an instantaneous event. The **mode** transition will occur over an interval of time during which some **threads** complete and undergo finalization and other **threads** undergo initialization and become ready for*

*dispatching. The timing and source and destination of message **connections** may vary during the **mode** transition interval.*

ACVIP planners and model developers should identify rules to limit the complexity of AADL **mode** declarations as needed for the planned analyses. Consider the capabilities of planned tools to perform multi-mode analysis. Limit the size and complexity of **mode** state machines (number of modes, number of transitions). Limit the extent to which different **mode** transition diagrams within different components interact with each other. Limit the extent to which declarations are mode-dependent. Where complex behaviors that are specific to a particular component need to be specified, features of the AADL Behavior Annex may be preferable.

3.7.4 Alternative Functional Behaviors

The term "functional mode" will be used in this handbook to refer to alternative sets of functional capabilities that can be provided by the system to its operators [27]. Functional modes would be described in operator manuals, for example. Care must be taken to distinguish this from AADL **system operating modes**, which refer to run-time architectural configurations within the system itself. AADL operating modes may be one language feature used to model functional modes, but functional modes may also be modeled using other kinds of AADL behavioral modeling declarations.

The term "component state" will be used in this handbook to refer to the internal data state of components, such as the values stored in memory at any point in time for a software component. The AADL Behavior Annex defines language features that should be used to model internal component discrete states and state changes [26].

Common measures of dependability are reliability, availability, and integrity [28] [29]. To enable analysis of these metrics, behaviors such as faults and conditions such as erroneous or failed must be modeled. The AADL Error Modeling Annex defines language features that should be used to model these kinds of behaviors [30].

High-level requirements such as those for functional modes and fault management may result in an architecture model that uses a combination of these three (AADL system operating modes, AADL Behavior Annex, AADL Error Modeling Annex). As with operating modes, ACVIP planners and model developers should identify rules to limit the complexity of models that mix these language features so the planned analyses are tractable. Consider the capabilities of available tools to perform such analysis. Limit the size and complexity of individual behavior declarations. Limit the interactions between behaviors declared in different behavioral modeling sub-languages.

3.8 Identify Change and Configuration Management Procedures

In an acquisition program that involves multiple models and organizations, different organizations are likely to use different processes and tools to name, store, and manage versions and configurations of models, analysis and test results, and other associated data. ACVIP plans should address processes and methods for common naming, versioning, and configuration management to use when exchanging model information between involved organizations. Configuration management may also need to be applied to

selected modeling and development environment tools and equipment. This section identifies situations likely to be encountered and issues that should be considered.

ACVIP plans should take advantage of existing methods and tools used for software source code change and configuration management where suitable, as discussed in Modularize Model Text and Diagrams.

In a virtual integration process, model information will be produced and consumed by different organizations. This may be accomplished in a number of ways. It may be accomplished by delivery of models, or by using a shared model repository, or by using a model server that provides controlled access to the model data needed for a specified purpose. The combination used will depend on the circumstances of each project.

ACVIP plans to produce and manage traceability, conformance, and consistency data should be considered when identifying methods for naming, accessing and managing shared model data and the tools used to create and analyze the model. Stakeholder needs for information discovery should be considered. The naming, versioning, and configuration management methods must support the development and verification of required traceability, conformance, and consistency data.

> *Example: The ACVIP plan states that the "single source of truth" mass data for to-be-developed equipment in AADL CDR models shall be taken from the mass properties analysis of a solid model of that equipment. The version naming and configuration management methods identified should assure that mass data appearing in a given version of an AADL model is the same as that produced by mass properties analysis of the appropriate version of the corresponding solid model.*

There should be common processes and methods to unambiguously name models, model versions, and configurations of single and multiple models, across all organizations involved in the development or use of a common set of models. The tools and environment that the models were built and analyzed with may also need to be tracked. These common processes and methods should be identified at a level of detail sufficient for the acquisition program.

> *Example: Developers in an acquisition program are encouraged to maximize use of commercial catalog parts. The ACVIP plan calls for models of commercial parts to be named and versioned as described in the commercial catalogs.*

> *Example: Two organizations will collaboratively develop a model using an agile development process. One organization will host a shared repository, shared configuration management system, and shared issue and task tracking system. Changes to common files will be managed using optimistic conflict resolution methods. Individual members of the teams will make direct contact with each other as desired, but all substantive exchanges should be captured in the shared issue and task tracking system to support project management and post-project process improvement studies.*

Configuration management may need to comply with other standards or regulatory requirements.

Example: Configuration management obligations are identified in RTCA DO-178C. A Software Configuration Index is one of the three always-required deliverables identified in that handbook.

Models may contain restricted information. Change and configuration management plans should take into consideration the issues discussed in Address Access Restrictions.

3.9 Plan Virtual Integrations

Virtual integration is an activity that requires input models, tools, skilled personnel, and time.

A major goal of ACVIP is to detect defects early. The ACVIP Management Plan should anticipate that delivered models will not successfully virtually integrate and pass all planned analyses at the first attempt. Virtual integration should be planned as a collaborative debugging task that is led by the system model integrator and supported by the component model suppliers. Schedules and the availability of technical resources should be coordinated and aligned.

ACVIP planners should consider methods and tools to support distributed collaborative engineering. For example, models may be exchanged between a model procurer and a model supplier using a shared repository.

Example: A system integrator provides controlled access to a common repository by all component model suppliers. The ACVIP Management Plan identifies software engineering practices familiar to all participants that are to be applied to support collaborative virtual integration, such as policies for branch management and merge conflict resolution. The AADL textual format is used, and conventional tools (originally developed for software source code management) are applied to manage collaborative integration and debugging of the AADL models.

The system architect should first develop a model of the overall system architecture at a high level of abstraction. This initial model should minimally identify all subsystems and components for which more detailed models will be acquired and virtually integrated. This AADL model will be derived from yet higher-level requirements, such as end user functional requirements and AADL patterns and libraries for family-of-systems architectures as discussed in Integrate with the Overall Development Process.

The following goals should be considered when planning component model definition, component model acquisition, and system model virtual integration.

- Model descriptions provided to model suppliers by a virtual integrator should reuse portions of the system model into which the procured models will be virtually integrated, e.g., descriptions of models to be procured are self-contained subsets of the system model.
- A model description provided to a supplier should require little or no modification by that supplier in order to make local use of that model, e.g., to use it as a virtual analysis harness.
- A component model delivered by a supplier should require little or no modification in order to be virtually integrated into the system model, e.g., it is a reusable model for a reusable component.
- A clear distinction should be made between properties that are requirements given to the model supplier and values to be provided by the supplier. Properties intended to be modified by the

integrator (such as binding declarations to integrate a component) should be identified and grouped together.

These are goals in the sense they are unlikely to be fully achieved. ACVIP plans should identify expectations and policies for model integrators and suppliers to change descriptive and delivered models where this is needed to accomplish virtual integration. An overarching guideline is to structure description models to reflect the overall project plan and supply chain structure. A recommended default policy is that the model integrator is responsible for model changes needed to accomplish virtual integration. This minimizes the need for complex coordination between multiple organizations and facilitates reusable models for reusable components.

Delivered component models may require modification by the model integrator. For example, dependencies may exist on mock environment models that were used during component model development. These dependencies may need to be changed by the model integrator. The model integrator may need to add additional data after receiving the component models. For example, component configuration **properties**, and **connections** and **bindings** between components, may be needed. Where possible, models should be structured so that **extensions** and **refinements** of the earlier specification or delivered component models can be used to declare modifications.

The virtual integrator will apply a specific set of tools to perform the planned analyses on the virtually integrated model. It is not necessarily the case that model suppliers will have all the tools used by the virtual integrator. In such cases, ACVIP planners should consider how component model suppliers will support the finding and fixing of defects in the virtually integrated model.

Model suppliers may have analysis tools that are not available to the model integrator. In some cases, this can be managed using the compositional analysis approach described in Address Access Restrictions.

4. Support Certification Approvals and Readiness Reviews

Projects must undergo a number of certifications. The checklist cited in the Defense Acquisition Guidebook has 26 potential certifications [31]. There are also a number of readiness reviews that could be supported by model review and analysis.

Failure to receive certification or readiness approvals is an important category of potentially expensive rework. ACVIP can be applied for the purposes of reducing defects that are not detected until late phase review by certification authorities identify shortcomings. ACVIP Plans should align with planned certification and readiness reviews. Specifications for analyses that address safety and security should align with certification procedures in order to reduce that category of defects and rework.

If ACVIP outputs are also submitted as evidence to certification authorities, a much higher level of assurance is needed that the analysis results are correct and that the models accurately describe the as-built system. The threshold is no longer that error detection effectiveness is good enough to significantly reduce cost, schedule and risk. ACVIP planners should evaluate which modeling & analysis activities can

also serve that purpose. Certification authorities require certain kinds of evidence in certain formats. Tools must be qualified. The models must be validated, see Assure System Conforms to Models.

4.1 DoD System Safety Process

MIL-STD-882E System Safety is the overarching framework for system safety in DoD programs [32]. MIL-STD-882E defines a system safety process that enables identification and management of hazards and their associated risks during system development and sustaining engineering, illustrated in Figure 7 Elements of the MIL-STD-882E System Safety Process.

The use of modeling and analysis as evidence to certification authorities is not necessarily a planned purpose of ACVIP, but planned ACVIP modeling and analysis activities should still align with safety processes in order to reduce project risk and rework due to problems found during certification. This section overviews the safety process to provide context for supporting analyses discussed in subsequent review sections.

MIL-STD-882E does not identify specific technical methods that should be used to accomplish elements of the system safety process. Specific safety objectives and methods are described in the Program System Engineering Plan and the project System Engineering Management Plan. Projects have their own tailored safety plans. ACVIP planners should determine which safety analyses should be performed based on project ACVIP goals and technical needs. The guidelines in this handbook are presented in the context of the following more specific safety processes.

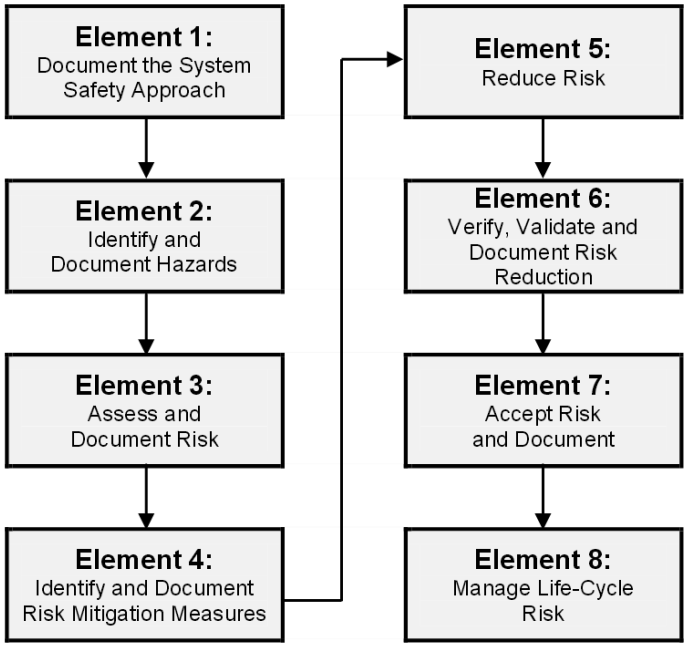

Figure 7 Elements of the MIL-STD-882E System Safety Process

4.1.1 SAE ARP4761 Safety Assessment Process

MIL-HDBK-516B Airworthiness Certification Criteria cites elements of *SAE ARP4761 Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment*, which

recommends specific analyses at specific phases of a system safety process [33]. Figure 8 SAE ARP4761 Safety Assessment Process Analyses illustrates analyses that may be performed to support this process and the directions in which information and traceability flow between analyses. Among these, this handbook provides guidelines for:

- Functional Hazard Assessment
- Markov Analysis
- Fault Tree Analysis
- Failure Modes and Effects Analysis

Preliminary System Safety Assessment and System Safety Assessment can be addressed using this handbook by viewing them as elaborations of Functional Hazard Assessment combined with traceability that shows how more detailed analyses provide evidence that risks have been satisfactorily mitigated.

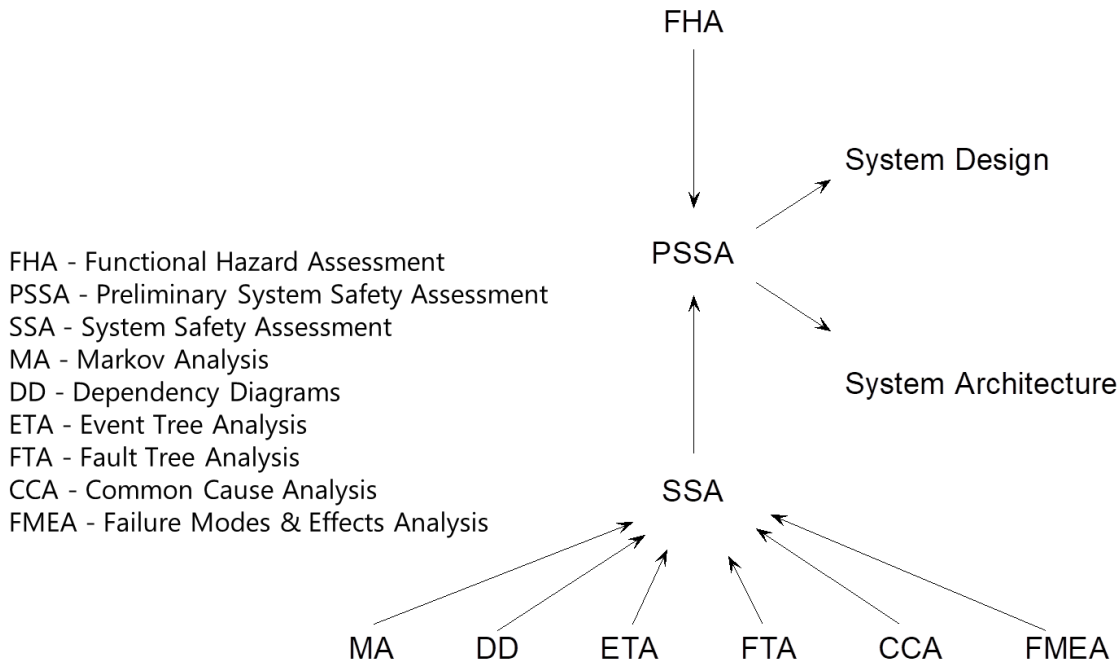

Figure 8 SAE ARP4761 Safety Assessment Process Analyses

4.1.2 System-Theoretic Process Analysis

System-Theoretic Process Analysis (STPA) is a hazard analysis technique for surfacing scenarios that lead to identified hazards and accidents [34] [35] [36]. STPA supports safety analysis from a systems-theoretic view of causality. The STPA approach views a system as control loops with nodes acting as sensors, controllers, actuators, and the controlled plant. Sensor and control signals pass between these nodes. STPA can identify a larger set of potential accident causes, including causes that do not involve component failures. STPA can identify hazards due to design flaws or unexpected interactions among otherwise operational components. STPA also considers influences outside the engineered system. STPA analysis can take into consideration human interactions, processes, and organizational structures that surround the system.

The STPA process begins with the establishment of the foundational elements of the system being analyzed. The first foundational element is the set of accidents and hazards for the system. In STPA an accident is defined as an event leading to loss. A hazard is defined as a set of system states that, when combined with worst case environmental conditions, will lead to an accident. The second foundational element is a set of constraints that will prevent the hazards from leading to accidents. The final foundational element is the top-level safety control structure of the system. This control structure can be modeled in AADL, identifying the STPA role (sensor, actuator or controller) for components and their interactions with each other. As part of the STPA analysis the developer applies risk controls to mitigate the hazards.

With the foundational elements identified, the STPA methodology shows how to analyze control loops in two steps to determine if inappropriate control actions, or lack of necessary control actions, can lead to accidents. The methodology identifies which conditions within the operation of the control loop components and which external factors can lead to hazardous control actions. STPA Step 1 uses guidewords to help identify unsafe control actions. These guidewords specify when a control action is:

1. Provided when not appropriate
2. Not provided when needed
3. Applied too long
4. Stopped too soon
5. Provided early
6. Provided late

Unsafe control actions are identified by applying the guidewords to all of the control signals within the system and tracking which ones can lead to hazards. Step 2 of STPA determines how these hazardous actions can occur within a system. This step looks at operations within a component such as inadequate control, or an inconsistent process model within a controller, or inadequate operation of a sensor or actuator. This step requires domain experts to identify scenarios where hazardous actions can occur even if no components fail.

Portions of the STPA analysis can be supported using an AADL system model. The control system can be modeled using AADL components. The control signals can be modeled with flows. The EMV2 error library can be leveraged to apply errors based on guidewords. Hazards and accidents can be represented using properties. The SRR Functional Hazard Assessment and other more detailed analyses that support the overall safety process contain guidelines for conducting an STPA process using AADL.

4.1.3 Airworthiness Qualification and Approval

DoD Directive 5030.61, *DoD Airworthiness Policy*, states that "all aircraft and air systems owned, leased, operated, used, designed, or modified by DoD must have completed an airworthiness assessment [37]." Each department establishes an airworthiness authority responsible for defining and overseeing an airworthiness qualification process and issuing approvals to operate.

Airworthiness Qualification processes require that an acceptable plan for system safety be developed early in the acquisition program. *DoD Directive 5030.61 DoD Airworthiness Policy* and its cited *MIL-STD-*

882E System Safety and *MIL-HDBK-516B Airworthiness Certification Criteria* provide guidelines for a system safety program plan. Each DoD department has its own set of more detailed directives and guidelines for airworthiness qualification.

- Army regulations require that "Army aviators and unmanned aircraft system operators will not operate aircraft in the performance of official duties if there is no airworthiness release or airworthiness approval [38]." The Combat Capabilities Development Command (CCDEVCOM) Aviation & Missile Center (AMC) Systems Readiness Directorate (SRD) is the delegated airworthiness authority for US Army aircraft [39]. SRD specifies and applies the Army Military Airworthiness Certification Criteria (AMACC) [39]. For each program, an Airworthiness Qualification Plan (AQP) will be issued by the customer and a responding Airworthiness Qualification Specification (AQS) will be provided by the supplier. Where ACVIP outputs are to be used as model-based evidence to support airworthiness qualification, the ACVIP Management Plan should align with these documents, and the Airworthiness Qualification Specification should cite the ACVIP methods and results to be used to support airworthiness qualification and approval.
- *(Citations for Navy airworthiness deferred to a later version.)*
- *(Citations for Air Force airworthiness deferred to a later version.)*
- *(Citations for NASA airworthiness and spaceworthiness deferred to a later version)*

RTCA DO-178C calls for the creation of a Plan for Software Aspects of Certification (PSAC) by a software developer. RTCA DO-254 similarly calls for the creation of a Plan for Hardware Aspects of Certification (PHAC). There are usually many software and hardware developers and PSACs and PHACs in a large acquisition program. In the context of these guidelines, the SRR model establishes system safety requirements that flow down to software and hardware components. ACVIP plans for architecture aspects of certification should provide guidelines for software and hardware developers to align their plans for certification with ACVIP plans, especially the model-based aspects of their plans for software and hardware component development and certification.

Wherever a PSAC or PHAC uses ACVIP modeling and analysis to satisfy an airworthiness qualification obligation, *RTCA DO-330 Software Tool Qualification Considerations* may be used to determine which AADL tools need to be qualified. ACVIP plans should identify such tools and the means for qualifying them using that guideline.

4.2 Security Assessment and Authorization

Three DoD Instructions specify processes to be used for cybersecurity assessment and authorization.

1. DoDI 8540.01, *Cross Domain Policy* [40], specifies the process for qualifying DoD Information Systems that must process classified information. This Instruction mandates the use of a Cross Domain Solution (CDS) where needed to isolate information at different security levels. This Instruction impacts a system architecture by requiring that all cross-domain information flows within the system pass through an approved CDS. This results in a system architecture that follows the Multiple Independent Levels of Security (MILS) approach of system design.
2. DoDI 8510.01, *Risk Management Framework (RMF) for DoD Information Technology (IT)* [41], specifies a six step process for categorizing the system in terms of its loss impact for information

Confidentiality, Integrity, and Availability, then selecting, implementing, and assessing security controls to mitigate those impacts, and finally approving the system and monitoring it for future problems.

3. DoDI 8500.01, *Cybersecurity* [42], highlights the need for Operational Resilience (OR). OR has three goals: to make information and information services always available to authorized users, to ensure that the system's security posture is always visible to system owners, and to enable the system to respond and recover with little or no human intervention. The system requirement for a Cyber Survivability Endorsement (CSE) addresses these and other goals.

DoDI 8500.01 Cybersecurity provides an overview of the security process for DoD Information Technology (IT) systems [42]. The use of modeling and analysis as evidence to security certification authorities is not necessarily a planned purpose of ACVIP, but planned ACVIP modeling and analysis activities should still align with security processes in order to reduce project risk and rework due to problems found during certification. This section overviews the security process to provide context for specific supporting analyses discussed in subsequent review sections.

Department of Defense Instructions (DoDI) define a hierarchy of requirements that impact security qualification for National Security Systems (NSS) like Future Vertical Lift (FVL). The umbrella instruction, *DoDI 8500.01 Cybersecurity*, summarizes the key areas for concern (see Figure 9). This Handbook focuses on two of those areas: security qualification for DoD Information Technology (IT) that process multiple levels of classified information in support of mission partners, and risk-based security qualification for all DoD IT. The following sections guide model developers to create supporting evidence, at major system development milestones, for each qualification. In the spirit of ACVIP, the guidance focuses on activities performed prior to CDR.

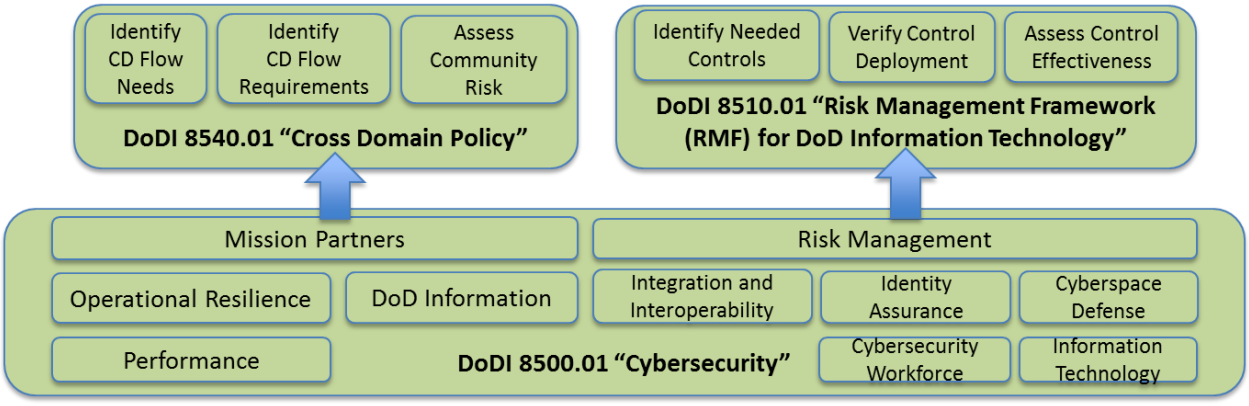

Figure 9. Two Key Cybersecurity Regulations for DoD IT

4.2.1 Cross Domain Policy

DoDI 8540.01 Cross Domain Policy requires DoD IT that will process multiple levels of classified information to use an approved cross domain solution (CDS) for information sharing between different security domains [40]. An approved CDS is one selected from the Unified Cross Domain Services Management Office (UCDSMO) Baseline List of Approved Solutions. Approving authorities pose three key

questions: is the CDS needed, what are its requirements, and what is the risk to the DoD community? Model-based engineering activities at SRR should demonstrate the need for the CDS, activities at PDR should identify the requirements that support the selection of the CDS, and activities at CDR should assess that selection within the overall system architecture to support an overall risk assessment. Model-based architectural analysis should minimize the risk that the system will fail an assessment against DoDI 8540.01.

A first step for the modeling activity is to identify the system boundary and those entities in the system's environment that either consume or provide classified information for the system. The system's environment determines the need for the CDS according to the external networks to which the system connects and the external users who access the system.

If the system's environment requires different levels of information security processing, then model analysis must demonstrate that the system isolates information security processing by security level. Specifically, analysis must show that the CDS partitions information processing within the system by security level. Model analysis should examine both explicit information flows, such as connections to external networks and users, and implicit information flows, such as bindings between software and hardware. Only model components designed as a CDS should observe information flows at multiple information security levels.

Systems with MLS processing requirements generally adopt a Multiple Independent Levels of Security (MILS) architecture, that is, single level components connect to a CDS to enable cross domain information sharing. MILS architectures provide rigorous separation between components at different levels, which is achieved either by physically separating these components on different execution platforms or by hosting the components on an access type CDS, which isolates its processing partitions. Information sharing occurs via a transfer type CDS, which converts information at one security level to a different security level.

4.2.2 Risk Management Framework Policy

DoDI 8510.01 Risk Management Framework (RMF) for DoD Information Technology requires a risk assessment of DoD IT for information assurance gaps [9] [41] [43]. Following the six-step RMF process (illustrated in Figure 10 Risk Management Framework Process Steps), the DoD IT system owner categorizes the system according to its impact on mission assurance given a loss of information Confidentiality, Integrity and Availability, selects security controls to minimize those losses, implements the security controls, assesses that implementation, and after approval to deploy the DoD IT, continues to monitor the system for potential losses. Systems with higher impacts implement stronger security controls. Model-based engineering activities at SRR should support system categorization, activities at PDR should support security control selection, and activities at CDR should support security control assessment.

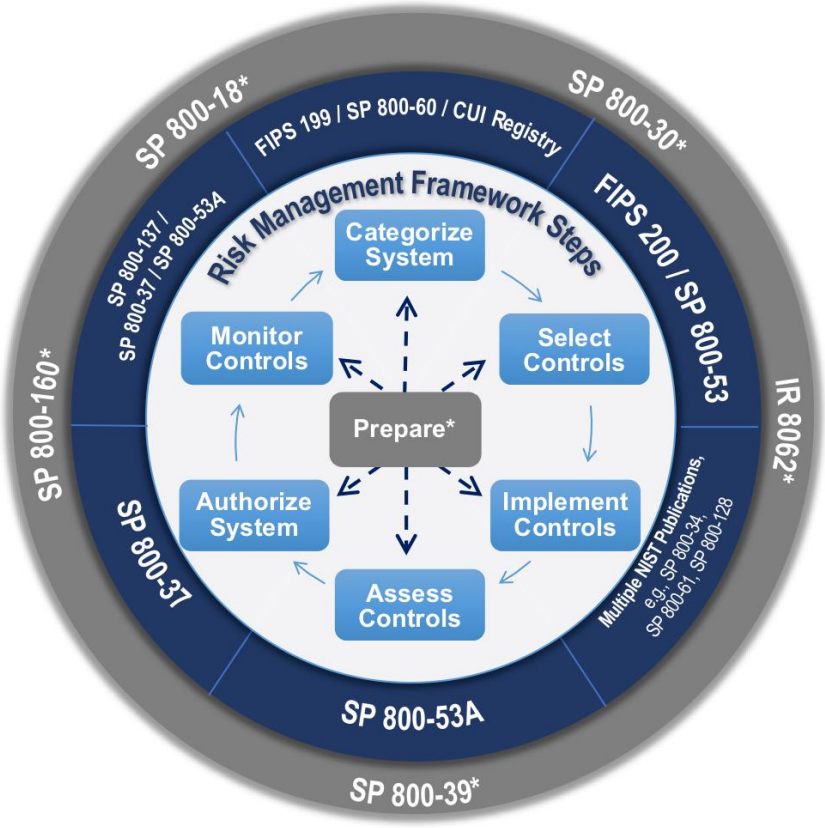

Figure 10 Risk Management Framework Process Steps

The first step is to model the system boundary. The model developer should model the entities in the system's environment that interact with the system, the information flows to and from those entities, and the impact of each information flow on the system's mission given a loss of Confidentiality, Integrity, and Availability (CIA). The system's environment determines the criticality of the information flows. The higher the impact of information loss, the greater the number of security controls required to protect that information flow. Model analysis should ensure that system users cannot use less critical flows to access or impact more critical flows.

Next, the model developer should model the security controls required to protect the information flows according to their CIA impacts. While the RMF process calls for choosing security controls based on the highest identified impact, choosing controls at the granularity of an individual information flow lets the model developer leverage architectures that isolate flows and avoid the need for security controls everywhere. This consideration is especially important for embedded systems that partition processing across space and time.

RMF security controls offer protection through policy and technical means. The model developer will focus on technical controls, that is, controls to be implemented in software and hardware. The model developer should specify the system components that will implement required technical controls. Model analysis should confirm that the system implements all technical controls required to fully protect each information flow according to its CIA impacts.

The RMF security control assessment (Step 4) measures the effectiveness of each control. The RMF process measures effectiveness largely in terms of whether the control operates correctly. Without an implementation however, it is difficult to measure correctness. So instead, model analysis at this stage should examine architectural considerations for effectiveness. For example, model analysis should assess whether or not it is possible to bypass the control and still access the protected information flow. Model analysis should also look for ways to tamper with the control's configuration and change the enforcement behavior of the control. These early, architecture-centric analyses support the RMF process's assessment of control effectiveness and help lower the risk of a failed assessment.

4.2.3 Cybersecurity Assessment and Approval

Each DoD service establishes its own processes for cybersecurity certification and approval.

- Army Regulation 25-2 Army Cybersecurity defines the role of Approval Officers (AO) who can issue an Approval To Operate (ATO) [44].

4.3 Physical Configuration Audit

The Physical Configuration Audit (PCA) is a formal examination to verify the "to be fielded" configuration of a validated system against its design and manufacturing documentation [45]. It is recommended that PCA include an examination to verify that the "to be fielded" system conforms to all delivered AADL models that specify and describe that system. The guidelines in Assure System Conforms to Models should be considered.

5. Define Model Content Needed for Analyses and Reviews

The primary way in which model content is determined in an ACVIP Management Plan is to specify the analyses that are to be performed on that model. The information that must be captured in a model is that which is needed to perform the required analyses with the required precision and certainty. Guidelines for developing analysis specifications and model content are given in Model and Analysis Precision and Uncertainty, Mixed-Fidelity Modeling and Analysis, and subsections on individual analyses.

This section is structured as a menu of analyses to be considered at familiar review milestones: SRR, SFR, PDR, CDR. These are used as well-known guideposts for the purpose of structuring this section. ACVIP itself does not recommend any specific set of reviews. Reviews may vary depending on the acquisition path and development process. Continuous virtual integration may be done in an agile or DevSecOps development process.

The models used at successive reviews should be elaborations of those used in preceding reviews, as discussed in Abstraction, Elaboration, and Conformance. Once an analysis is introduced at a review, it should be rerun for subsequent reviews. The precision and certainty of analysis results will increase at successive reviews due to the use of increasingly elaborated and verified models.

Some basic kinds of analysis are introduced in the subsection on SRR, the review where goals and requirements for selected analyses are first specified. Subsequent subsections will include additional analysis specification guidelines as needed. This list is not exhaustive. ACVIP planners should consider what tools and analyses are available to them. The guidelines for analysis specifications cite review criteria and products listed for these major milestones in the Defense Acquisition Guidelines, Chapter 3, Systems Engineering [45]. ACVIP planners should consider these in addition to project-specific goals and requirements.

This handbook is tool-agnostic. These guidelines use conventional terms for kinds of analyses and cite standard AADL features to use for model content. The ACVIP Management Plan is likely to identify kinds of analyses as is done in this handbook. Detailed analysis specifications for specific projects will identify specific tools, details of model content, and details of analysis results and their implications.

It is not possible to exactly and completely capture to-be system behavior using AADL semantics and language features. The selected tool assumptions and behaviors may not exactly match the technologies and detailed design patterns selected for the to-be system. When developing analysis specifications, whether during planning or execution, differences between tool assumptions and behavior and the technologies and design patterns selected for the to-be system should be evaluated. The correspondence between tools and system technologies does not need to be exact, but the analysis results need to be acceptable for the selected purpose. Where significant differences exist between tool assumptions and system technologies, that should be considered as part of risk and uncertainty management.

> *Example: A selected timing analysis tool assumes that a message will be sent by a thread at the completion of each execution of that thread (the instants a thread is suspended awaiting the next dispatch). In the selected Real-Time Operating System (RTOS), a message is sent by a thread by calling a send service at any point during its execution. The system will exhibit a greater range of message send times than is assumed by the analysis tool. The ACVIP plan includes a task for engineers performing timing analysis to review analysis assumptions of the tools and identify any uncertainty or error that might be introduced into the analysis results. For example, greater jitter in message send times may increase anomalous scheduling effects in some multi-resource systems* [46] [47].

Most analysis tools operate on an instantiation of a specific **system implementation** declaration. Model comments and model user guides should identify the **system implementation** declaration to be selected for each analysis.

5.1 Model and Analysis Precision and Uncertainty

Models can vary widely in their level of detail and uncertainty. The level of detail has a significant impact on the ability to detect defects and assess and manage risk – and the cost and schedule required for modeling & analysis. This also affects how work is divided among different organizations, since the role of one organization is often to receive a model having modest detail and then deliver back a model that adds significantly more detail. This section introduces concepts and guidelines to describe what level of detail should be provided in a model.

Terms such as “functional,” “logical,” and “physical” are often used informally to characterize the intended use or level of abstraction of a model. Some may associate specific meanings and processes with these terms [48]. Some terms are defined in standards with technical meanings specific to those standards, such as “computation independent,” “platform independent” and “platform specific” in the Object Management Group (OMG) Model Driven Architecture (MDA) standards or the “conceptual,” “logical,” and “platform” levels in the Future Airborne Capability Environment (FACE) data modeling standard. Where models are being delivered and integrated, the term “level” has also become associated with level or tier in the hierarchy of the model structure (e.g., system is level 1, subsystem is level 2). The term “level” (and “tier”) is also sometimes used when discussing the structure of the supply chain. Terms like these can be convenient ways to quickly and roughly indicate the level of detail of a model. However, they may also connote process phases or model purpose. This handbook does not define or recommend any particular set of labels to intuitively characterize the information content of a model. Where ACVIP planners choose to use such terms, they should be careful that their meaning is made clear in the context of a specific project and performing organizations.

This handbook uses precision and uncertainty to further characterize the detail needed in a model for a specific purpose. Precision refers to the degree of refinement with which an analysis is performed or a measurement stated. It is the amount of information and level of detail in the model and its analysis results. Model precision is not the same thing as model accuracy, or the degree to which the model and its analysis results accurately describe the final system. This topic is discussed in Assure System Conforms to Models.

Uncertainty refers to the degree of trust that the values produced by an analysis are close enough to the actual values for the desired purpose. Precision and uncertainty in model property values and precision and uncertainty in analysis results are different things. The latter must be determined as a function of the former using methods such as uncertainty propagation and sensitivity analysis.

To decide or describe more exactly what needs to be captured in a model, the recommended method is again to work backwards from the desired analysis. What level of precision and certainty is required in the analysis results?

The exact characterization of precision depends on the type of analysis. Identifying the analysis needed is the first step in describing the needed model content – it is the content needed to run the required analysis. However, many analysis tools will adapt to the amount of information in the model – they provide results for what is declared in the model. The level of decomposition of a system into subsystems and sub-subsystems may need to be described in the plan or in the analysis specification. The AADL **categories** of components that are included in a model may need to be specified.

> *Example: A model with AADL **system** declarations for software and hardware with resource supply and demand properties such as MIPS and BPS is sufficient for initial resource loading analysis. When these **systems** are elaborated to specific **processors** and **threads**, the model has sufficient precision to do resource-loaded schedule (schedulability) analysis.*

Analysis tools compute values from the information in the model. Uncertainties in the analysis results depend on uncertainties in the parameters of the model. In order to assess this, the analysis method and tool must do some form of sensitivity analysis or uncertainty propagation [13]. Depending on the analysis, this may require that uncertainties be captured in the model in a suitable way.

> *Example: A virtually integrated model includes two redundant sensors, a compute module, and a display. The compute module and display hardware are existing components whose failure rates are well-known. The sensors are a new product whose fault rate has been estimated. The ACVIP Management Plan calls for a fault tree analysis to be performed at PDR, where the specified analysis tool will output overall function reliability and also importance and sensitivity analysis values. Importance analysis helps identify fault events that contribute most to the system's unavailability. Sensitivity analysis helps identify fault events where a relatively small change in a fault rate will lead to relatively large changes in function reliability.*

ACVIP planners should consider what support for sensitivity analysis and uncertainty propagation is available in analysis tools, and how much additional modeling effort is needed to determine and capture parameter uncertainties in the model. This is particularly important when a goal is to reduce risk as discussed in Reduce Technical Risk.

5.2 Mixed-Fidelity Modeling and Analysis

Different subsystems in a model may have different degrees of precision and uncertainty. This is conventionally called mixed-fidelity modeling. This can easily occur in a model created by virtually integrating other models or in a model where some components have yet to be fully specified. Models for legacy components may only exist as black-box models. ACVIP planners should consider what level of detail is needed in the different parts of a model, and in the parameters for different kinds of analyses, in order to achieve the desired benefits with the least modeling effort. It is expected that most complex system models will be mixed-fidelity.

> *Example: A system integrator is virtually integrating a model of a sensor, three models of software components that process sensor data, and a model of a display. The three software component models are virtually integrated into a compute module. The sensor, software, and display communicate over a switched network. ACVIP planners are most concerned about the timing and loading of the compute module. The ACVIP Management Plan says the sensor and display models integrated for PDR may consist only of AADL **type** declarations that declare message contents and transmission rates and internal latency upper bounds. The three software component models shall include AADL **implementation** declarations that specify threads and message hand-shaking protocols. A latency timing analysis tool is selected that is able to determine end-to-end latency bounds using black-box sensor and display subsystem models and a white-box compute module subsystem model.*

Some decisions about which parts of a model should be elaborated with more detail and which are at a sufficient level of detail might best be made during execution rather than initial planning. Planners should

consider the use of uncertainty and sensitivity analysis methods to guide decisions made during program execution (rather than during program planning) about the details of an analysis specification.

Mixed-fidelity modeling and Abstraction, Elaboration, and Conformance relations within evolving models create uncertainties in whether or not a model contains all the information needed for a particular purpose. There are standard AADL properties to control legality rule checking by tools for **classifier** matching and **signature** matching. However, in general tools decide what feed-back is provided in terms of errors, warnings, uncertainty and sensitivity data in reports, etc. Different tools have different policies about what they will assume as default values for undeclared properties. Whether or not particular errors or warnings are of concern depends on the tool and purpose. Large and complex models may result in large numbers of warnings and large amounts of analysis data. ACVIP planners should consider what tool support is available and methods used to configure and triage tool feed-back.

5.3 System Requirements Review

The System Requirements Review (SRR) ensures that all requirements are defined and consistent with cost, schedule, risk, and other system constraints; and with end user expectations. One class of requirements is system performance requirements derived from the Initial Capabilities Document (ICD) or draft Capability Development Document (CDD). There are requirements derived from regulations and policies such as the safety and security policies discussed in Support Certification Approvals and Readiness Reviews**.** There are requirements derived from key business drivers such as compliance with standards and architectural patterns to satisfy MOSA mandates.

Subsections for each analysis will open by citing Defense Acquisition Guidance (DAG) review criteria that may be addressed by that analysis [45]. Planners should consider additional criteria that apply to individual projects. The following DAG SRR criteria apply broadly to all analyses.

- Technical risks are identified, and mitigation plans are in place.
- Contractor clearly demonstrates an understanding of the system requirements consistent with the ICD and draft CDD.
- System requirements are assessed to be verifiable.
- Requirements can be met given the plans for technology maturation.
- Key technology elements have been identified, readiness assessed and maturation plans developed.
- Program technical risks are adequately identified and documented such that there is a clear understanding regarding the contractor's ability to meet the specification requirements.
- Draft verification methodologies have been adequately defined for each specification requirement.

5.3.1 SRR General Guidelines

The primary ACVIP SRR activity is to assure requirements that are to be satisfied or verified by ACVIP are adequately captured in a combination of the ACVIP Management Plan and an initial requirements model, including preliminary specifications of the analyses to perform during the project. SRR determines that the ACVIP Management Plan, if properly executed, will satisfy and verify the requirements and goals

allocated to ACVIP activities. The ACVIP model should capture all key interfaces, business drivers, performance requirements, etc., that are to be verified by analysis or model-based testing at some point during the project. The SRR model should be traceable back to customer models and documents used to specify their requirements.

To the maximum extent possible, requirements should be captured in analyzable model declarations such as **types**, **flows**, and **properties** rather than natural language "shall" statements.

Requirements are often captured in a mix of natural language documents and SysML. Where requirements are captured in SysML that are to be satisfied or verified by ACVIP activities, an AADL Profile and supporting tool should be applied to those model elements so those requirements are automatically synchronized with their AADL representations and tracked during virtual integration.

SRR is also the time at which preliminary analysis specifications are reviewed. ACVIP Management Plans will minimally identify categories of requirements and goals to be addressed by different kinds of analyses. At SRR, analysis specifications should be sufficiently detailed to judge whether an analysis will satisfy or verify a requirement. Analysis specifications will at some point include details such as identifying specific tools to be used, but details may be deferred where rolling planning is done.

Analysis results are not limited to simple property, value pairs. Analysis results often include visualizations such as graphs and curves. Some analysis results are explored using interactive tools. Model-based review procedures should be adapted as needed for non-traditional forms of requirements specification and analysis results.

The model should include a modularization of the system architecture into major subsystems and components as needed to specify required key interfaces. The system boundary is a key interface that should be modeled. The Defense Acquisition Guidebook SRR products and criteria checklist includes a preliminary identification of all software components, which may require further detail in how the system is modularized into components [45].

System-level requirements trace to more detailed derived requirements. For example, an end-to-end latency requirement may trace to derived latency requirements for subsystems. Such allocations often change during development, e.g., one supplier provides a model at PDR showing they are well within their memory allotment while another is over. Plans should accommodate changes in how requirements are allocated to derived subsystems and components. Tools should be selected that support trade studies and uncertainty management.

Applying analysis tools to the SRR model is expected to produce limited results. The SRR model primarily serves to establish requirements for subsequent modeling and analysis. Some analyses may issue errors if run on the SRR model because those requirements have not yet been satisfied in that model. Analysis results that flag errors can be useful to assess whether the captured requirements are adequate. For example, Multiple Independent Levels of Security (MILS) requirements may reasonably result in errors because there are no Cross Domain Solutions (CDS) in the model yet. Examination of the MILS analysis

results that identify security level conflicts is useful to validate those requirements. Removal of all analysis errors is not necessary at intermediate review milestones, only that all of them be addressed in some way.

Additional content should be added as needed to guide the development of models for subsequent reviews. Additional content should be added as needed to define architectural alternatives and support trade studies. Additional content should be added as needed to support project and technical risk management. The SRR model serves as the initial specification for the more elaborate models that are to be subsequently developed, procured, and virtually integrated. The guidelines provided in Describe Models to be Developed and Delivered should be applied to the SRR model to flow-down requirements for component models and component ACVIP activities.

5.3.2 SRR Technical Plans Review

Several plans are reviewed at SRR [45]. The ACVIP Management Plan should be consistent with and support other plans. Here are examples of plans that are likely to be related to ACVIP plans.

- System Engineering Management Plan
- Modeling & Simulation plans, including the ACVIP Management Plan
- Risk Management plans
- Configuration Management plans
- Test plans
- Certification plans

5.3.3 SRR Bidirectional Traceability Established

Bidirectional requirements traceability should be established by SRR [45]. Traceability goes through multiple kinds of assets. Traceability should be established between ACVIP plans and other plans, between documents and models, between the AADL and other models, and within individual models.

The SRR model should declare requirements that are allocated to the architecture and its components and are to be verified by analysis of the architecture model as discussed in SRR General Guidelines. The SRR model should establish modeling patterns and conventions that will be used to elaborate traceability information as the project progresses through subsequent reviews.

Traceability must be done across different kinds of model-to-model boundaries.

- Traceability must be established between high-level stakeholder requirements in formats such as natural language documents, systems engineering modeling languages (such as SysML), and the AADL models.
- Requirements must be traced between AADL component models and other formats and assets used for detailed component specifications, such as FACE USMs and test specifications and results.
- Requirements must be traced from the AADL models to analysis results obtained by applying tools to those models.

Traceability modeling guidelines should be identified in the ACVIP plans. Common guidelines for representing traceability analysis are recommended to make it easier for different stakeholders to review models from different suppliers, and traceability guidelines are likely to be identified in the government Program ACVIP Plan. In practice traceability is done using several mechanisms in different modeling languages. Navigation through traceability is often facilitated by tooling. Tools may infer some traceability relationships from explicitly declared data in models, flag unexpected presence or absence of traceability links, and assist managing requirements changes. The traceability modeling guidelines for a project should be adapted as needed to handle traceability to, from, and within AADL models.

AADL **extends** and **refines** declarations can be used within an AADL model to implicitly establish traceability for some requirements and analyses without additional special traceability declarations. For example, SRR Interface Static Consistency Analysis can make use of this form of implicit traceability. Guidelines for controlling **property** value inheritance are given in Abstraction, Elaboration, and Conformance. Whether or not property inheritance is sufficient to establish traceability from an earlier to a more elaborate model depends on the kind of analysis and tools that make use of those properties.

> *Example: The customer requirements included a performance requirement that the crew be alerted of a particular class of threats within 1 second of a sensor detecting those threats. This is captured in a **Latency property association** for an **end to end flow** from the sensor to the cockpit display. The SRR model shows this flow passing through a sequence of **flow path specifications** declared in the **types** (interfaces) of major subsystems. As the SRR model is elaborated into PDR then CDR models, **flow path implementations** are declared for these **flow path specifications** as part of the **implementations** of the subsystems. The AADL language features for flows implicitly capture traceability from the high-level **end to end flow** in the SRR model to a fully implemented and detailed **end to end flow** in the CDR model.*
>
> *Example: The draft AADL Requirements Definition and Analysis Language (RDAL) Annex provides a proposed standard way to capture additional requirements information and explicitly declare traceability links in an AADL model. Tools may be available that support draft annexes.*
>
> *Example: The Object Management Group (OMG) Requirements Interchange Format (ReqIF™) standard or Open Services Lifecycle Collaboration (OSLC) Requirements Management standard could be used to interface with external requirements data bases.*
>
> *Example: A contractor receives stakeholder requirements including a SysML model. This model consists primarily of requirements, use case, and activity diagrams. There are a few block definition and implementation diagrams that show the system boundary and key interfaces to government-furnished software and hardware components. The contractor uses a SysML AADL Profile and translation tool to generate AADL **system** declarations from SysML block diagrams, including translation of selected parametric constraints into AADL **property** and **annex** declarations. Requirements traceability that occurs within the SysML model stays in the SysML model. Traceability from SysML to AADL is implicit in the well-defined and deterministic translator mapping that preserves naming and references SysML element UUIDs. All manually created*

*portions of the AADL SRR model are declared as **extensions** and **refinements** of the generated portions so that re-generation can be used to re-establish traceability without over-writing hand-written AADL modifications. Where AADL language features and analysis tools do not establish implicit traceability for a class of requirements and analyses within the AADL model, the contractor uses a tool and property set based on the draft AADL RDAL Annex.*

5.3.4 SRR Modeling and Human Inspection

Human inspection is a cost-effective way to detect many categories of defects [12]. An organization should adapt its chosen model inspection techniques to apply to AADL architecture models. Planners should consider defect categories and scenarios that are may not be detected by planned automated analysis using tools. Appropriate allocation of requirements for and on an AADL model (e.g., conform to customer modeling guidelines, do virtual integration to assess MOSA update scenarios) will likely require human inspection. Reviewers should consider which goals and requirements have been appropriately allocated to ACVIP activities, which ones should have been but were not, which ones were but should not have been.

Model creation should be considered a form of analysis or structured requirements elicitation. The act of creating a specification in a rigorous modeling language can detect defects. This can reveal gaps, ambiguities, and inconsistencies detected by basic language legality rule checking. Planners should consider categories of defects that are likely to be detected by model creation, even if analysis tools are not applied to that category. Identification of key interfaces (e.g., interfaces selected to meet MOSA goals) is an example of a requirement that can be satisfied by identifying that interface in the model. Tools can later be applied to analyze criteria such as consistency with other interfaces and compliance with required standards.

Reviewers should evaluate model content, analysis specifications, and available analysis results as part of technical risk assessment. If something seems overly complicated in the model, or there are concerns about how well the model and analysis specifications align with anticipated detailed design and implementation methods, that should be assessed for technical risk.

Example: The ACVIP plans call for human inspection of models to assess whether technical risks are identified in the models and mitigations are included in the system design. Models are reviewed for content for life-cycle support, training devices, tactics, mission system, etc.

5.3.5 SRR Interface Static Consistency Analysis

External interfaces to the system should have been documented. Preliminary identification of all software components should have been completed. System requirements are sufficiently detailed and understood to enable functional definition and functional decomposition. [45]

An architecture is defined to be a set of components and the interactions between them. Some form of interface static consistency analysis can be applied to any architecture viewpoint.

Static interface consistency analysis is a set of checks on the static structure of the model. Interfaces are declared using AADL **type** declarations. **Features** of a type should be present to specify information

provided or required by a component or dependencies it has on other components. Various specialized forms of **feature** declarations provide information about information exchange mechanisms, e.g., information can be transferred using messages, shared objects, or subprogram calls. To analyze the consistency between two interfaces, **connection** declarations are used to identify dependencies between specific interfaces. **Binding** declarations are used to identify resource dependencies, such as a software component dependency to be hosted on an operating system that complies with a specified standard.

There are a variety of static interface consistency checks and tools that could be selected. There are scripting languages and tools designed to easily tailor consistency checks, analogous to software bug-finder tools. The SRR models and analysis specifications need not necessary contain all these details, but they should have sufficient detail to specify what should be run at some point to satisfy or verify requirements.

Example: The OSATE AADL Integrated Development Environment performs model legality rule checks defined in the AADL standard. This includes checking type-compatibility between the two ends of a connection.

Example: A tool that verifies, for each data item required by a component, there exists at least one provider of that data in the system.

Example: A tool that verifies the binding of a connection through a sequence of platform resources is consistent with the hardware connections and categories of those resources.

Example: A tool that verifies the FACE execution profile declared as a property of a software component is consistent with the FACE execution profile declared by the resource to which that software component is bound.

Example: A tool that uses an AADL annex sublanguage to declare pattern rules and traverses the model to verify those rules are satisfied.

5.3.6 SRR Interface Behavior Consistency Analysis

Interface behavior consistency analysis can be applied to functional, software, and hardware viewpoints.

There are currently several standard ways to specify run-time behavior in AADL models.

- AADL **properties** of **features** can be used to select among a discrete set of communication protocols and their properties. Static interface consistency checks can verify that communication protocols are consistent between the communicating components.
- AADL Behavior Annex declarations are used to declare state transition systems with guards and actions, where transitions can be triggered by various kinds of events. Functional behaviors can be declared in either **types** or **implementations** for any **category** of component. This is the primary language feature that should be used to model general functional behavior.

- AADL **system operating modes** are used to declare architectural reconfigurations that occur at run-time. Both **types** and **implementations** may declare **modes** and **transitions** between **modes** that are triggered by run-time events. Most AADL declarations have an **in modes** clause that allow users to say that **properties** may have different values in different **operating modes** or that different sets of **connections** and **threads** are active in different operating modes. Identify Variation Points, Configurations, and Dynamic Behaviors provides guidelines to distinguish cases where AADL **modes** should be used versus other mechanisms.
- AADL Error Modeling Annex declarations are used to declare fault, error and failure behaviors in components and architectures. Guidelines for these features are provided in SRR Reliability, Availability, and Failure Analysis and SRR Functional Hazard Assessment.
- SysML interactions, displayed using sequence diagrams, are considered a kind of behavior. These map to AADL **flows**. AADL **flows** are used in many kinds of analyses, and these are discussed separately with each analysis rather than in this subsection.

Behavioral consistency analysis checks a model to see if a set of properties is true for the composition of multiple component behaviors. Behavior analysis verifies behavioral assertions over the system state space, such as assertions that can be expressed in a temporal logic. The specific kinds of behavioral assertions that can be verified, and the way in which those assertions are declared, will depend on the selected tool.

Behavior analysis is an example of an analysis where there is a wide range of choices to make for model content and tools. It is only practical to model selected behaviors with selected degrees of precision and uncertainty. Select Cost-Effective Modeling & Analysis provides guidelines for making such decisions.

> *Example: Behavior annex declarations are used to specify a state machine that models the life cycle of a component. At SRR, the identified component states are Stopped, Initializing, Running, Finalizing. The plan calls for connections to be modeled, and the set of life cycle states refined, at PDR for the exchange of events between components that would cause changes in component life cycle states. A tool is selected to analyze the system of interacting state machines to determine if there exists any order in which components can be initialized that results in failure to provide data needed for other components to complete their initialization. This behavior was selected for modeling because previous projects had experienced intermittent deadlock during system start-up and shut-down, the root cause defect was difficult to track down and fix, and little additional model content needs to be added to the core architecture model to enable this analysis.*

5.3.7 SRR Resource Loading Analysis

The DAG SRR criteria include determining that software functionality is consistent with software sizing estimates and resource loading. Model content and preliminary analysis of resource loading using budgets or estimates for demands and supplies should be performed sufficient to satisfy that criterion.

Resource loading analysis is typically applied to a combination of software and hardware viewpoints, including standard AADL binding **properties** between the two viewpoints.

Resource loading analysis uses properties that declare demands and supplies for various kinds of resources, where some components can supply certain kinds of resource units that other components demand. The AADL core standard defines processor capacity (Millions of Instructions Per Second), communication bandwidth (Bits Per Second), and memory (Bytes) resource units. Tools can support additional kinds of units, such as Power and Mass used for Size, Weight, and Power (SWaP) assessment. AADL binding declarations are used to associate demands with supplies, e.g., a software component with a MIPS demand is bound to a processor that can supply MIPS.

> *Example: The system integrator uses data from previous projects to estimate software demands for Millions of Instructions Per Second (MIPS) and byes of memory; and to estimate execution environment supplies for those resources. These are declared as **properties** of software and hardware **systems**, **processes**, and **processors**. **Binding** declarations capture preliminary allocations of software to hardware. Thresholds (budgets) are established that will cause analysis tools to issue warnings and errors. A utilization analysis tool is specified that will provide sensitivity data so that technical risk due to uncertainty can be tracked. The initial analysis results are reviewed at SRR to show that software functionality is consistent with software sizing estimates and resource loading.*

> *Example: A system must process significant amounts of complex sensor data. Some of this is done using Field-Programmable Gate Arrays (FPGA), and Configurable Logic Blocks (CLB) are important resource units to manage. Data from previous projects that is used to make initial estimates for network loading was recorded in Frames per Second (FPS) rather than Bits per Second (BPS). The system integrator, in their analysis specification, identifies a tool that allows the user to define their own resource units for resource loading analysis.*

5.3.8 SRR Latency Analysis

Latency analysis provides information about the timing of data that is flowing through multiple components in a system. Data flows through a system that have latency requirements should be declared in the model using AADL **flow** declarations.

Latency analysis can be applied to functional, software, and hardware viewpoints where information flows are modeled.

AADL **flow** declarations (a sequence of **data** and **event flow connections** between components) should be used to specify end-to-end dependencies between information arriving at one **flow source** of the system boundary and departing at another **flow sink** of the system boundary. These declare end-to-end (system input to system output) flows of data and events through the system.

The AADL standard **Latency** property should be used to specify end-to-end latency and jitter requirements on **flows**. This specifies the interval of time between when information arrives to the system and when it affects the information leaving the system (departing information may also depend on other information flows and internal state). A standard **Actual_Latency** property is available for measured latency when such data becomes available.

AADL allows a decomposition of a **system end-to-end flow** into sub-flows through **flow paths** of capability components. Sub-flow latency and throughput properties establish derived timing requirements (budgets) for subsystems and components.

How time is managed in a system is a fundamental architectural decision, e.g., globally synchronous, globally asynchronous, globally asynchronous locally synchronous (GALS). If the time frame-of-reference for all component input and output events is not global Newtonian time, then AADL features to declare synchronization domains and their properties and scopes should be used to identify the different time frames of reference for different subsystems and interfaces between time domains.

A simple form of latency analysis is to check consistency between end-to-end flow requirements and sub-flow requirements derived from them. Analysis that verifies consistency between system latency requirements and derived/allocated subsystem latency requirements may be desired at SRR.

> *Example: An information flow from sensor to display has the requirement that sense-to-display (end-to-end) latency shall not exceed 1000ms. During requirements analysis developers decide to decompose this into derived timing requirements: (1) the sub-flow latency through the sensing function shall not exceed 200ms; (2) the sub-flow latency through the fusion and tracking function shall not exceed 600ms; and (3) the sub-flow latency through the cockpit display function shall not exceed 250ms. An analysis of the model reveals that the sum of the sub-flow latencies exceeds the maximum allowed sense-to-display latency.*

Throughput metrics used in timing requirements, such as sampling rates and messages-per-second, should be clearly defined. **Property sets** for throughput metrics should be declared as needed when standard AADL **properties** are not defined. **Properties** that relate incoming and outgoing throughput metrics with other system capacity requirements should be defined where needed. **Properties** to specify behavior under overload and failure conditions should be defined where needed.

> *Example: A Situation Awareness (SA) system has several sensors that provide object detections at various rates measured in detections-per-second. The SA system also has a requirement that it manage a minimum number of fused tracks for the combined incoming events. The ACVIP plans call for a set of properties to be declared in an AADL **property set** together with a technical specification of throughput and capacity metrics in terms of these properties.*
>
> *Note: Average throughput rate and peak burst rate are different metrics. For example, there might be a steady-state requirement to process 10 object detections-per-second and a burst requirement to process 200 object detections in a 10 second interval that is preceded and followed by 10 second intervals having no more than 20 detections. Burst requirements are sometimes specified as a minimum number that shall be queued without loss for a specified steady-state throughput assuming a given inter-arrival time distribution. Queuing protocol **properties** can be used to specify overload behavior, e.g., in the above example sensors might be prioritized so that data is discarded from low-priority sensors rather than from high-priority sensors during overload conditions.*

5.3.9 SRR Reliability, Availability, and Failure Analysis

SRR is an opportunity to assess whether technical requirements for Reliability, Availability, and Maintainability are flowed to specifications, although the DAG does not call for analysis until PDR. [45] There are several kinds of analysis that can be applies in this area. This handbook will discuss PDR Failure Modes and Effects Analysis, PDR Fault Tree Analysis, PDR Reliability Block Analysis, and PDR Markov Analysis in later sections.

Reliability, availability, and failure analyses can be applied to functional, software, and hardware viewpoints.

The SRR model should capture reliability and availability requirements using features of the AADL Error Modeling Annex. Only the features of the annex and details of the model necessary to capture requirements are needed at SRR. Additional model content can be added later to enable the selected analyses.

Figure 11 AADL Error Modeling Annex basic concepts illustrates some key concepts and terms that will be used in this handbook. Each component in a model may have an **error model** associated with it using an annex declaration. An error model takes the form of a state machine, where states and events may be typed and transitions may have both stochastic and discrete semantics. A transition between error states may be triggered, for example, by a random internal event such as a hardware fault or an external discrete event such as an AADL **system operating mode** change.

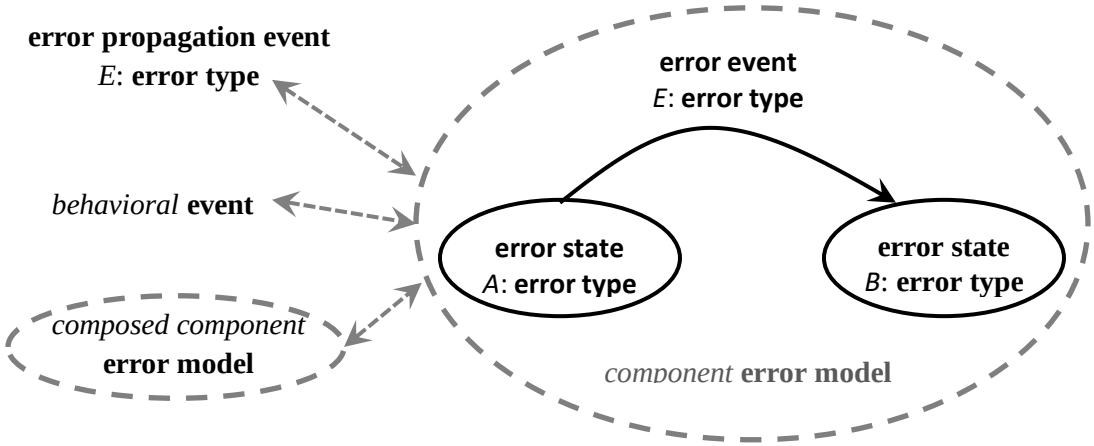

Figure 11 AADL Error Modeling Annex basic concepts

Different error models for different components may interact with each other using three primary mechanisms. Error events that occur in one error model may propagate to another error model according to the structure of the AADL model, for example through an AADL **connection** between the two components. Discrete behavioral AADL **events** may also propagate into or out of an error model according to AADL **event connections**. Finally, an error model for a component may be declared as a composition of the error models of its subcomponents, which defines the error states of the component as a function of the error states of its subcomponents.

The AADL Error Modeling Annex provides features that can be used for a variety of different analyses. The exact interpretation of these features depends on the specific analysis. For example, an AADL error state can model a hazard state, a latent fault state, an internal error state, a failure state, or a state in which the component is undergoing repair, depending on the analysis tool and purpose of the model.

The AADL Error Annex includes an **ErrorLibrary** package that contains a set of pre-declared **error types** and a few simple pre-declared **error models**. Elements from this standard **ErrorLibrary** should be preferred where suitable. ACVIP planners should consider the exact interpretations and capabilities of available tools to determine detailed guidelines.

Example: Figure 12 Example Reliability Requirement Declared using Error Modeling Annex Features illustrates how reliability requirements can be specified in an SRR model.

```
-- The probability of failure during an 8 hour mission
-- shall not exceed 1.0E-4.
annex EMV2 {**
    use behavior ErrorLibrary::FailStop;
    properties
        EMV2::OccurrenceDistribution =>
                [ ProbabilityValue => 1.0E-4; Distribution => fixed; ]
             applies to FailStop;
        EMV2::ExposurePeriod => 8.0 applies to Operational; -- Time in hours
**} ;
```

Figure 12 Example Reliability Requirement Declared using Error Modeling Annex Features

Example: Figure 13 Example Availability Requirement Declared using Error Modeling Annex Features illustrates how availability requirements can be declared in an SRR model.

```
-- The fraction of the fleet that is available on the flight line for
-- immediate dispatch at any point in time shall not be less than 90%.
      annex EMV2 {**
             use behavior ErrorLibrary::FailAndRecover;
             properties
                 EMV2::OccurrenceDistribution =>
                     [ ProbabilityValue => 0.9; Distribution => fixed; ]
                  applies to Operational;
      **} ;
```

Figure 13 Example Availability Requirement Declared using Error Modeling Annex Features

5.3.10 SRR Functional Hazard Assessment

Hazards should have been reviewed and mitigating courses of action should have been allocated within the overall system design. Certification requirements should be understood [45].

DoD System Safety Process provides an overview of the DoD safety process. Even if the ACVIP Management Plan does not call for modeling and analysis to be used as evidence for certification authorities, modeling and analysis activities should align with required certifications in order to reduce

project risk and rework due to problems found during certification reviews. Additional guidelines when modeling and analysis are to be submitted as evidence are found in Support Certification Approvals and Readiness Reviews.

Functional Hazard Assessment (FHA) is a safety assessment performed for the overall system and its intended operations and environment of use [33]. FHA establishes the overarching system safety technical requirements for the system. FHA or its equivalent is required by system safety standards (e.g., Identify and Document Hazards in Figure 7).

Functional hazard assessment is applied to a functional viewpoint. This initial assessment is used to derive safety requirements to be satisfied or verified in data, software, and hardware architecture viewpoints. Analyses can be applied at these later levels-of-refinement to verify that identified functional hazards are acceptably mitigated.

The **EMV2 property set** defined in the AADL Error Modeling Annex declares a **Hazard property**. This property has a record type that can record several pieces of information about a hazard, including severity, likelihood, risk, and design assurance level. The **Hazard** property can be associated with any category of Error Modeling Annex feature. This handbook suggests that error states be defined to represent hazards, with information about each hazard state declared using a **Hazard** property, but the conventions of the selected tool will take precedence.

An Error Annex **property set MILSTD882** declares constants for **Severity** that range from **Negligible** to **Catastrophic**. The **MILSTD882** property set declares constants for **Likelihood** that range from **Frequent** to **Improbable**. A safety policy defines a method for determining risk as a function of severity and likelihood and establishes acceptable risk thresholds based on that determination. A tool can determine if there is any combination of severity and likelihood that violates a given risk assessment and acceptance policy.

The SRR model should include an initial allocation of mitigating courses of action needed to reduce risks to acceptable levels. The SRR model should capture in some way any mitigations introduced for each hazard state. These are derived safety requirements that should appear in the SRR model. The guidelines provided in Abstraction, Elaboration, and Conformance and SRR Bidirectional Traceability Established may be used for this purpose.

> *Example: A Situation Awareness (SA) system has several sensors that provide object detections. The functional hazard assessment identifies a failure to advise the crew of obstacles as hazardous. A severity of 1 is assigned. A likelihood of D is assigned due to an included sensor fusion capability that will reduce false positives.*
>
> *Example: Figure 14 Risk Assessment Matrix from MIL-STD-882E (from MIL-STD-882E Department of Defense Standard Practice for System Safety) illustrates how qualitative assessments of SEVERITY and PROBABILITY (likelihood) are combined to assess risk. DoDI 5000.02 Operation of the Defense Acquisition System requires "that the associated risks have been accepted by the*

following acceptance authorities: the [Component Acquisition Executive] CAE for high risks, Program Executive Officer-level for serious risks, and the Program Manager for medium and low risks [49]*."*

RISK ASSESSMENT MATRIX				
SEVERITY / PROBABILITY	Catastrophic (1)	Critical (2)	Marginal (3)	Negligible (4)
Frequent (A)	High	High	Serious	Medium
Probable (B)	High	High	Serious	Medium
Occasional (C)	High	Serious	Medium	Low
Remote (D)	Serious	Medium	Medium	Low
Improbable (E)	Medium	Medium	Medium	Low
Eliminated (F)	Eliminated			

Figure 14 Risk Assessment Matrix from MIL-STD-882E

5.3.11 SRR Cross Domain Analysis

Security Assessment and Authorization provided an overview of the DoD cybersecurity process. Even if the ACVIP Management Plan does not call for modeling and analysis to be used as evidence for certification authorities, modeling and analysis activities should align with required certifications in order to reduce project risk and rework due to problems found during certification. Additional guidelines when modeling and analysis are to be submitted as evidence are found in Support Certification Approvals and Readiness Reviews.

Cross domain analysis applies to the data viewpoint and to flows of information between elements of functional, software, and hardware viewpoints.

Model developers should define the system boundary using one of the techniques described in Describe Models to be Developed and Delivered and SRR General Guidelines. The model should include all entities outside the system boundary, i.e., in the system's environment, that interact with the system and that serve as a consumer or provider of classified information. While these entities could represent human users, they are more likely to represent the devices with which these users interact with the system. The model developer should keep in mind that while the user may be privileged to view information at multiple levels of security simultaneously, that user is more likely to view each level on a separate device, so the model should include each device.

Next the model developer should associate with each external entity the highest security level of information to be transferred between that entity and the system. The model developer may create this association using an AADL property or by other means.

At the level of detail of an SRR, the system may be a black box and model analysis may simply issue errors that requirements are not satisfied by the model. The objective for SRR is to determine the need for a cross domain solution (CDS) given the system's environment. In a different environment, analysis using the same system may produce different results.

> *Example: The mission system on an air vehicle platform processes information received from sensors and radio networks and also interacts with passengers in the cabin, who are cleared at a lower security level than the cockpit crew. With no other details about the system implementation, analysis confirms that the mission system observes information at multiple levels of security. The conflicts identified in the analysis results provide information about requirements for cross domain solutions.*

MILS analysis is an example of an analysis that may not be performed because the system has no MILS requirements.

5.3.12 SRR Risk Management Framework Analysis

Programming architectures and security requirements should have been identified. Certification requirements should be understood.

RMF analysis applies to the data viewpoint and to any kind of flow of information or events between elements of functional, software, and hardware viewpoints.

Model developers should define the system boundary using one of the techniques described in Describe Models to be Developed and Delivered and SRR General Guidelines. The model should include an abstract representation of external consumers and providers that communicate with the system over this boundary.

Next the model developer should associate with each external consumer or provider the information flows between that entity and the system, and for every information flow, specify in the model the mission impacts given a loss of Confidentiality, Integrity, and Availability (CIA) of that information flow. The model developer may create these associations using an AADL property or by other means. This activity supports Step 1 of the RMF, "Categorize the System".

At the level of detail of an SRR, the system may be a black box. The objective for SRR is to determine the CIA impacts for information flows processed by the system as required by the system's environment. Applying analysis tools to determine if appropriate security controls are included where required within an architecture and cannot be bypassed or tampered with may only produce error messages at SRR.

5.3.13 SRR Trade Studies Requirements

System performance specification has sufficiently conservative requirements to allow for design trade space. Risk management plans should be in place and technical risks identified. The system performance

specification should be adequately expanded to reflect tailored, derived and correlated design requirements. Product support plan and sustainment concepts should have been defined with the corresponding metrics. [45] Meeting these criteria can be addressed by identifying architectural alternatives and performing trade studies.

Trade studies may involve multiple kinds of analysis. The architecture viewpoints and their content will depend on the set of analyses chosen for a trade study.

The guidelines for Configurable Models and Variation Points should be used to model architectural alternatives to be evaluated during trade studies. This establishes at SRR an initial design space to be explored during future trade studies.

Ideally an analysis tool is available for each key performance parameter (quality metric) to be assessed during a trade study. Where qualitative or other manual assessments are performed for a metric, care should be taken that the metric may depend on the overall model configuration and not just a specific component. The guidelines for Configurable Models and Variation Points can be used to establish a naming convention for model configurations.

> *Example: A contract calls for a trade study of weight v. power v. reliability v. four alternative bundles of mission system functionality. Plans to conduct this trade study are to be reviewed at SRR. The SRR model captures alternatives for redundancy by declaring multiple **implementations** for major compute platform (execution environment) **subcomponents** of the system. Hardware components in these alternative implementations have weight and power **properties** and AADL Error Annex **error models** declared. The SRR model captures the alternative bundles of functionality by declaring multiple **implementations** for major software sub-**systems** of the overall mission **system**. The contractor uses a trade space exploration framework to automate trade studies as follows.*
>
> - *The framework recognizes multiple **implementations** for a **subcomponent type** as candidate variation points.*
> - *The framework recognizes **properties** as candidate variation points.*
> - *The framework allows users to interactively select subsets of **implementations** and subsets of **properties** and their ranges to define the architectural design space to be explored.*
> - *The framework automatically enumerates configurations within the selected architectural design space.*
> - *For each configuration, the framework runs an automated software-to-hardware binding tool and then weight summing, power summing, and fault tree reliability analysis tools.*
> - *The framework provides a trade space visualization and exploration graphical interface that allows users to iteratively refine the scope and parameters of the trade study.*
> - *The trade study is re-executed as component models are updated by suppliers. The evolving results are available to contractor and customer personnel using a web interface.*

5.4 System Functional Review

The System Functional Review (SFR) establishes the functional baseline. This describes the system's performance (functional, interoperability and interface characteristics) and the verification required to demonstrate the achievement of those specified characteristics. The functional baseline is directly traceable to the operational requirements contained in the Initial Capabilities Document (ICD) and draft Capability Development Document (CDD).

The SFR criteria most relevant for ACVIP are to document allocated requirements optimized through analyses (including functional analysis and sensitivity analysis) and perform trade studies and risk assessments.

5.4.1 SFR General Guidelines

A functional architecture is a different viewpoint than a software or hardware architecture viewpoint. Layering, Extension, and Refinement provides guidelines on modeling a functional architecture viewpoint. That section also provides guidelines on associating functions with subsystems and components that were identified in the software and hardware architecture viewpoints.

A number of analyses can be applied to functional as well as software and hardware architecture viewpoints. There are no specific analyses cited in the DAG, but ACVIP planners should consider analyses that address goals and requirements. Planners should consider what functional analyses will be required at future milestones and identify model content that should be included in the SFR functional architecture in order to maintain traceability from SRR to PDR. Since minimal information will be present in SRR software and hardware architecture viewpoints, analyses should be considered that can be done independently of the software and hardware viewpoints.

5.4.2 SFR Interface Static Consistency Analysis

*Example: A functional architecture is developed that consists of **abstract type** declarations for functions whose **features** identify abstract types of information that may be needed as inputs or can be provided as outputs. Otherwise, the semantics of a function is defined by reference to a customer-provided ontology of functions. **Extends** and **refines** declarations are used to decompose a high-level function into multiple simpler functions. There are no connections or bindings. A tool is applied to verify that, for each kind of information needed by a function, that kind of information is provided from some **feature** found in the system interface or some other function in the system.*

5.4.3 SFR Interface Behavior Consistency Analysis

Example: A functional architecture is developed that uses a tool-specific Annex to declare assumptions (preconditions) a function makes about its inputs and guarantees (postconditions) it makes about its outputs when those assumptions are true. The inputs of functions are satisfied by connections from the system interface or outputs of other functions. A function will operate correctly if the assumptions declared on all of its inputs are satisfied by the guarantees declared on the sending side of its connections. A tool is

applied to verify that all assumptions made by each function is satisfied by guarantees made by other functions on which it depends.

5.4.4 SFR Failure Modes and Effects Analysis

Example: A functional architecture is developed that includes error annex modeling of functions and declarations of information dependencies between functions. A preliminary functional failure modes and effects analysis is performed to help assess whether functional mitigations in place reduce risk to an acceptable level.

5.5 Preliminary Design Review

The Preliminary Design Review (PDR) ensures the preliminary design and basic system architecture are complete. It provides the acquisition community, end user and other stakeholders with an opportunity to understand the trade studies conducted during the preliminary design, and thus confirm that design decisions are consistent with the user's performance and schedule needs and the validated Capability Development Document (CDD). The PDR establishes the allocated baseline. The allocated baseline describes the functional and interface requirements to a level in the system architecture sufficient to define hardware configuration item requirements distinct from software configuration item requirements, together with the verification required to demonstrate achievement of those requirements. [45] The following items from the PDR criteria apply generally to all PDR modeling and analysis.

- All risk assessments and risk mitigation plans have been updated, documented, formally addressed and implemented.
- Risks are at an acceptable level to continue with detailed design.
- Approach/Strategy for test and evaluation defined in the Test and Evaluation Master Plan (TEMP) accounts for risks with a mitigation plan; necessary integration and test resources are documented within the TEMP and current availability align with the program IMS. See Perform Model-Based Testing.
- Analysis of system performance is complete and assessed to meet requirements.
- Preliminary design (hardware and software), including interface descriptions, is complete and satisfies all requirements in the functional baseline.
- Computer system and software design/development approach have been confirmed through analyses, demonstrations, and prototyping in a relevant environment.

5.5.1 PDR General Guidelines

The PDR model will be an elaboration of the SRR model that fully identifies software and hardware configuration items and their interfaces. The guidelines in Abstraction, Elaboration, and Conformance and SRR Bidirectional Traceability Established should be followed to establish conformance between PDR and SRR models and analysis results.

A basic ACVIP guideline is to increase precision, reduce uncertainty, and maintain consistency between the model content and analyses going from SRR to SFR to PDR to CDR. Where an analysis listed in System

Requirements Review is not listed here, the recommendation is to review and refine those models and repeat those analyses with improved precision and certainty.

A PDR model may contain **process**, **subprogram group**, and **data** declarations (software objects); and **virtual processor**, **processor**, **virtual bus**, **bus**, **device**, and **memory** declarations (hardware objects). System objects that have no **subcomponents** may be used to model either software (if bound to something else) or hardware (if something is bound to them). **Abstract** objects should be reserved for objects in the environment of use, not in the system being acquired. **Subprograms**, **threads**, and **thread groups** should be modeled where this is needed for a planned structural analysis or when they are separate deliverables, but otherwise this may be unnecessary detail at the structural level of abstraction.

Many PDR analyses use information about which software components and connections are bound to which hardware components, either explicitly declared or automatically generated. Analysis results will be incomplete if binding declarations are incomplete, which will occur wherever binding decisions are to be made by the software and system integrator. Analysis results may be different for different possible bindings and for different possible modes of operation. Where software and system integrators are to make final binding decisions later in development, **allowed binding properties**, or **virtual processor** or **virtual bus** or **system** models of resources, may allow analysis to be performed at PDR that can then be elaborated at CDR.

5.5.2 PDR Interface Static Consistency Analysis

Computer system and software architecture designs should have been established. All Computer Software Configuration Items (CSCIs), Computer Software Components (CSCs), and Computer Software Units (CSUs) should have been defined. Software Requirements Specifications (SRSs) and Interface Requirement Specifications (IRSs), including verification plans, are complete and baselined for all CSCs and satisfy the functional requirements. Interface control documents trace all software interface requirements to the CSCIs and CSUs. All external interfaces to the system, and all internal interfaces of the system (system element to system element), as addressed at the SRR, should have been documented and modeled. [45]

Software components should be bound to hardware subsystems or components. Software components may be bound to virtual layers that are declared as **virtual buses** or **virtual processors** in AADL. Such virtual layers may also be bound to other virtual layers or physical resources such as **processors** and **buses**. These binding declarations specify the resources and execution environments for software components. Analyses verify that **properties** of software dependencies on hardware, such as use of specified operating system, are satisfied by software bindings to hardware.

> *Example: A family of systems will support software applications that use either the FACE™ 2.1 ARINC 653 safety profile or the FACE™ 3.0 POSIX security profile. The ACVIP plan calls for the creation of a property set that can be used to declare the execution environment required by a software component and provided by an AADL **virtual processor** or **processor**. A conformance checking tool that operates on a set of syntactic rules is selected. The ACVIP plan calls for the*

*creation of a set of rules that will verify consistency between the execution environment properties of a software component and the **virtual processor** or **processor** to which it is bound.*

5.5.3 Interface Behavior Consistency Analysis

At PDR, software and hardware components are identified. If connections over which events flow and preliminary behavior models exist, tools can be applied to verify whether or not a planned set of correctness assertions are satisfied.

Example: A refinement of the analysis specification at PDR selects a model-checking tool to verify that deadlock will not occur during initialization, restart, and shut-down during the life-cycle of any component for any possible sequence of triggering events.

Example: A refinement of the analysis specification at PDR selects a simulation tool to verify that a set of identified functional scenarios are correctly executed. The tool selection decision was in part based on a desire for a tool capability to perform user-guided simulation.

5.5.4 PDR Failure Modes and Effects Analysis

Failure Modes and Effects Analysis is one kind of analysis that can satisfy SRR Reliability, Availability, and Failure Analysis requirements.

Functional failure mode, effects, and criticality analysis (FMECA) should be completed [45]. Functional FMECA will require a functional architecture be modeled as discussed in SFR General Guidelines.

Failure Modes and Effects Analysis (FMEA) can be used to specify error-handling capabilities that are required to mitigate risks identified by hazard assessment. FMEA begins by identifying errors that may propagate into a system from the external environment and internal errors that may occur due to the nature of a component. Analysis of the model determines how these errors propagate from component to component given declared error-handling requirements. Analysis identifies errors and propagation paths that result in system failures.

FMEA is typically done in a bottom-up manner. System input errors and internal errors are first identified for the lowest-level components in the model. Analysis then propagates these through dependent components to system outputs. Where component models are obtained from suppliers, the models they deliver for virtual integration should include the necessary declarations.

AADL Error Model features should be used to declare errors that may occur within capabilities or propagate into or out of capabilities. AADL Error Model features exist to declare how a capability should respond to incoming and internal errors, for example by masking them or by outputting less severe errors.

ACVIP plans should include tasks to assure the consistency between FMEA analysis and any Fault Tree Analysis (FTA) or Reliability Block Diagram (RBD) analysis that is done. Failure modes and effects identified during FMEA should trace to basic events and faults in any FTA or RBD analysis that is done. This should be done using guidelines from Abstraction, Elaboration, and Conformance and SRR Bidirectional Traceability Established.

5.5.5 PDR Fault Tree Analysis

Fault Tree Analysis is one kind of analysis that can satisfy SRR Reliability, Availability, and Failure Analysis requirements. Fault tree declarations can be used to specify redundancy and independence-of-failure among different capabilities that are required to mitigate risks identified by hazard analysis.

> *Note: SAE ARP4761 Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment assumes a system safety process that starts with overall aircraft acquisition. An FTA analysis may be performed at the functional level, but an FTA analysis may also only be used for system hazard assessment after the aircraft system structure has been decomposed to a level of abstraction that corresponds more closely to the PDR model of this handbook. ACVIP planners should evaluate the level of abstraction at which FTA is first performed based on the system being acquired and its system safety plan.*

Fault tree analysis can be applied to functional, software, and hardware viewpoints.

> *Example: Functional hazard assessment (FHA) has identified runway excursion due to a failed capability to stop the aircraft as a hazard whose risk must be mitigated. Two redundant capabilities to stop the aircraft on the ground are specified, a wheel braking capability and a thrust reverse capability. The pilot must receive advance notice of brake system failure in order to properly apply thrust reverse as a back-up capability. Figure 15 Risk of runway excursion with redundant capabilities to stop aircraft shows the fault tree that specifies these requirements. Implicit in this specification is that there be independence-of-failure between wheel braking, notification to the pilot that the wheel brake system has failed, and thrust reverse capabilities.*

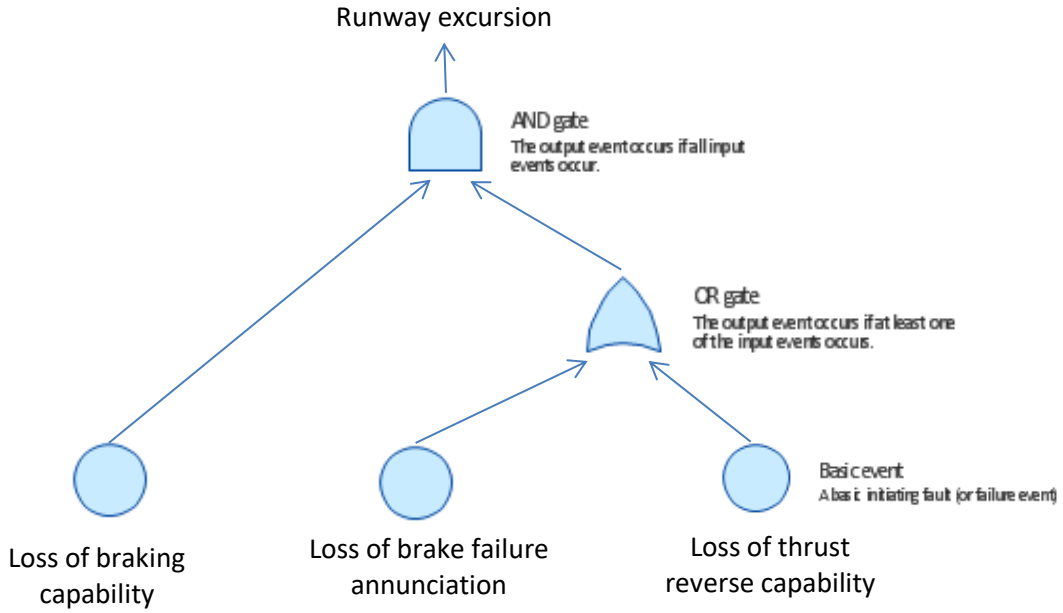

Figure 15 Risk of runway excursion with redundant capabilities to stop aircraft

The following paragraphs provide general guidelines for declaring models from which fault trees can be automatically generated. Detailed guidelines should be based on the selected tool.

Tools that perform FTA will generate and analyze one fault tree for each selected hazard. Recall from SRR Functional Hazard Assessment that an error model with hazard states is the recommended way to declare hazards. Such hazard error states can be designated as roots of fault trees to be generated and analyzed.

Error models are also declared for components that may undergo intrinsic failures, which is to say components associated with fault tree basic events. The initial **error state** for each component is an **operational state**. A fault tree basic event is represented by a transition into an **error state** that is designated as a **failed state**. Where quantitative analysis is to be done, properties declared for the transitions from operational to failed states are used to determine failure probabilities for basic events.

Redundancy in a model is typically declared using **composite error models** – for example, a system is declared to be in a failed **error state** when 2 **ormore** of its 3 redundant subcomponents are in a failed **error state**. Voting protocols within a component are typically declared using Error Model Annex conditional expressions on **error propagations** and **error transitions** in the **error model** for the component.

The structure of a fault tree itself can be automatically generated from this information based on all possible ways that errors might propagate within the architecture model, e.g., via connections or bindings or shared accesses. One such fault tree should be generated for each root hazard state identified during FHA and selected for fault tree analysis.

Qualitative analysis can be performed to identify single points of failure. Cut set analysis can identify, for each system failure, sets of fault tree basic events that will result in that system failure mode. Any system failure that has a cut set with only one element has a single point of failure, identified by the element in that cut set. These results can be checked against severity, likelihood, and safety policies identified during hazard analysis.

> *Example: The safety requirements for an air vehicle include the safety policy that all hazards that have a severity of Critical or Catastrophic must be mitigated by redundant capabilities for which there are no single points of failure. A cut set analysis of a fault tree generated from the AADL model is used to verify this.*

Where fault rates can be determined, a quantitative analysis can be performed to determine a probability of failure for each root of each fault tree. Importance and sensitivity analysis can be performed.

5.5.6 PDR Reliability Block Analysis

Reliability block analysis is one kind of analysis that can satisfy SRR Reliability, Availability, and Failure Analysis requirements. Estimate of system reliability and maintainability should be updated, based on engineering analyses, initial test results, or other sources of demonstrated reliability and maintainability.

RBD analysis can be applied to functional, software, and hardware viewpoints.

Reliability block analysis determines reliability for a capability based on the reliabilities of the other capabilities that it depends on and information about redundancy among those other capabilities. An RBD for a capability is often represented as a graphical AND/OR diagram as illustrated in Figure 16. RBD may be considered as an alternative to FTA. Traceability between hazards identified by FHA and capabilities whose failures contribute to those hazards should be captured.

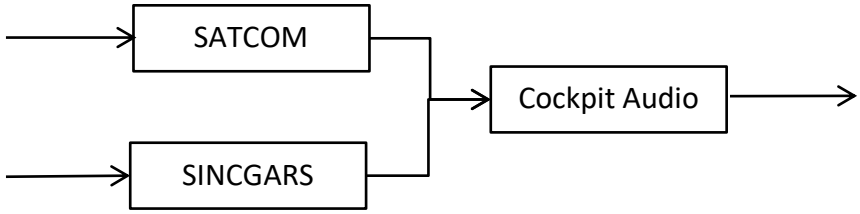

Figure 16 Capability fails if (SATCOM OR SINCGARS) AND Cockpit Audio fail

The AADL Error Annex provides features that should be used to declare **properties** and dependencies for RBD analysis. One approach is to use AADL **composite error model** features to declare a RBD structure for a system or capability in terms of sub-capabilities declared as AADL subcomponents. An AADL **composite error model** allows the **error states** of a component to be declared as an AND/OR function of the error model states of its subcomponents. RBDs and operational-versus-failure states for any sub-sub-capabilities are determined recursively.

> *Note: This idiom requires that capabilities on which multiple other capabilities depend be modeled as shared **subcomponents**. Modeling of shared sub-capabilities in higher-level RBD specifications may become complex when using this RBD modeling idiom.*
>
> *Example: A situation awareness system includes redundant sensors. The sensors are combined in a voter pattern so that when too many sensors fail, the sensing capability is considered failed. The RBD analysis will determine the reliability of the sensor configuration (the probability that the sensing capability will be available throughout a mission scenario).*

5.5.7 PDR Markov Analysis

Markov Analysis is one kind of analysis that can satisfy SRR Reliability, Availability, and Failure Analysis requirements. Markov analysis can be applied to functional, software, and hardware viewpoints.

FTA and RBD assume that a component begins each mission in a fully functional state. When a failure occurs, that component remains failed for the duration of the mission. In contrast, Markov analysis can be applied to systems that have degraded modes of operation, suffer transient errors, or can reconfigure and recover.

In system safety applications, continuous time Markov analysis is often used. Two forms of continuous-time Markov analysis can be applied, transient and steady-state.

1. Transient Markov analysis requires mission duration as an input. Transient analysis determines, for each error state a system of components might enter, the probability that it has entered a state at least once by the end of the mission scenario. Analysis can also determine values such as the expected number of times a system error state has been entered during the mission. Transient analysis is typically used to estimate probabilities for system failures or degraded modes of operation during a single mission.
2. Steady-state Markov analysis assumes the component error models have cyclic paths of transitions through every error state. The assumed mission duration is infinite. Steady-state Markov analysis determines the asymptotic probability of finding the system in a given system error state. Analysis can also determine values such as the mean time between visits to a system error state. Steady-state analysis is typically used to determine availability over an indefinite period of time, where the error models include models of maintenance and repair events as well as system error events.

The guidelines from PDR Fault Tree Analysis can be used for Markov analysis. Additional **error model** structures will be supported by a Markov analysis tool, such as **error models** that transition back-and-forth between operational and failed states.

Markov analysis can be computationally intensive and is subject to the state space explosion problem. Its use should be confined to high-level and simple models, or to analysis of individual components where such behaviors are important to estimate component error and failure probabilities that are then used in other more tractable analysis applied to the entire system. Tractable Markov analysis for a complex system model can be achieved by deriving from the complex model a simpler and more abstract one that captures the essential error behaviors. As with all abstractions, this incurs an obligation to assure that analysis results obtained from the simpler more abstract model are sufficiently accurate for the original more complex model.

5.5.8 PDR Cross Domain Analysis

If the model analysis at SRR indicates the need for a CDS, then prior to PDR or CDR the model developer should introduce the CDS into the model along with enough system subcomponent connection detail to show that the CDS effectively partitions the system architecture by security level. Repeat the model analysis performed at SRR to demonstrate that as the model developer adds details to the model, no system subcomponent, except the CDS itself, processes information flows for more than one security level.

An objective at this milestone is to support selection of the CDS from the Unified Cross Domain Services Management Office (UCDSMO) Baseline List of Approved Solutions. The model developer models the CDS as a black box since the CDS already exists, and model analysis should reveal the information security levels to be processed by the CDS. The model developer may wish to add more detail to support the selection decision, such as the message formats, the performance envelope, and power, size, and weight budgets.

5.5.9 PDR Risk Management Framework Analysis

By PDR or CDR, the model developer has created a software architecture to process the information flows identified at SRR. With these new details, model analysis should confirm that the software architecture does not mix information flows with different CIA impacts in the same process space. In particular, model analysis should confirm that if a software process contains subcomponents (e.g., threads) that process information flows of different criticalities, then all of these subcomponents must agree on the criticalities of the flows they process. This condition is necessary because while most operating systems guarantee isolation between software processes, they do not guarantee the isolation of subcomponents within a software process.

Given strong flow isolation within the software architecture confirmed, the model developer should next model the security controls required to protect each flow given that flow's CIA impacts. These activities complete Step 2 (Select the Controls) of the RMF process.

An objective at this milestone is to minimize the number of components that process highly critical information types by isolating, as much as practical, the information flows involving those types to a few components. By doing so, the model developer minimizes the number of required security controls and reduces the cost of implementation, testing, and assessment.

5.5.10 PDR Trade Studies

Trade studies and system producibility assessments should be under way. Software trade studies addressing commercial-off-the-shelf, reuse, and other software-related issues are completed. [45]

5.6 Critical Design Review

The Critical Design Review (CDR) provides the acquisition community with evidence that the system, down to the lowest system element level, has a reasonable expectation of satisfying the requirements of the system performance specification. The CDR establishes the initial product baseline for the system and its constituent system elements. It also establishes requirements and system interfaces for enabling system elements such as support equipment, training system, maintenance, and data systems. DAG criteria that are relevant to AADL modeling and analysis include (but are not limited to) [45]:

- Detailed design (hardware and software), including interface descriptions are complete and satisfy all requirements in the system functional baseline.
- Requirements tracing among functional, allocated, and initial product baselines is complete and consistent.
- Key product characteristics having the most impact on system performance, assembly, cost, reliability, and sustainment or Environment, Safety, and Occupational Health (ESOH) have been identified to support production decisions.
- Failure Mode, Effects, and Criticality Analysis (FMECA) is complete.
- Estimate of system reliability and maintainability based on engineering analyses, initial test results or other sources of demonstrated reliability and maintainability.
- Software functionality in the approved initial product baseline is consistent with the updated software metrics and resource-loaded schedule.

- Software and interface documents are sufficiently complete to support the review.
- Verification (Developmental Test and Evaluation (DT&E)) assessment to date is consistent with the product baseline and indicates the potential for test and evaluation success.
- All risk assessments and risk mitigation plans have been updated, documented, formally addressed, and implemented.

5.6.1 CDR General Guidelines

The CDR model fully captures the design architecture of the system. Any more elaborate detail is captured in models for individual components using modeling languages suitable for each component's application domain, e.g., Modelica, UML, VHDL. In keeping with the RTCA DO-331 distinction between "specification model" and "design model," the AADL models of component interfaces and key performance parameters are specification models, while the various models for component implementation details are the design models.

Model content for all previously performed analyses should be reviewed and updated to the most precise and certain information available for CDR. The analyses should be re-run.

5.6.2 CDR Resource-Loaded Schedule Analysis

This analysis determines if a specific set of **threads** and **connections** bound to specific **processors** and **buses** that use specific scheduling protocols satisfy a declared set of latency and throughput properties. Resource-loaded schedule analysis can provide component utilizations, upper and lower bounds on latencies and throughputs at points along event and data **flows**, and bounds on queue sizes and waiting times. This analysis can provide sensitivity analysis for **properties** in the model such as **Compute_Execution_Time**.

There are two general methods widely-used to perform this analysis, Monte-Carlo timing simulations and schedulability analysis.

Monte-Carlo simulation analysis uses an executable representation of the model to simulate multiple possible sequences of execution, keeping track of virtual time. The number and sequence of scenarios that are simulated are determined by parameters of the tool. Most such tools come with libraries that simulate common scheduling protocols or kinds of components. Most such tools allow additional protocols or components to be created using a scripting or conventional programming language.

Schedulability analysis uses mathematical methods to determine analytic upper and lower bounds on all possible behaviors admitted by the model. Different schedulability analysis algorithms are used for different scheduling protocols. Schedulability analysis tools can produce sensitivity analysis results.

The modeling and analysis effort required is typically less for schedulability than simulation because fewer details need to be modeled. Solution times may be faster due to the avoidance of large numbers of simulation runs. Schedulability analysis produces upper and lower bounds for all possible behaviors of the model, whereas simulation (like testing) only provides results for the scenarios that were executed. Simulation is more flexible, as it can analyze any protocol that can be written in software, while schedulability analysis is limited to algorithms that have been developed for specific kinds of scheduling

protocols. Schedulability analysis provides analytic bounds that may be pessimistic, and tightness of bounds should be assessed.

Analysis requires that threads and connections and their bindings to processors and buses be declared, together with a number of **properties** that declare **thread** and **connection** dispatch protocols and **processor** and **bus** scheduling disciplines. Shared **data** components and the protocols used by **threads** to access them must be declared. Bindings may be depicted in a layered architecture, and properties of **virtual processors** and **virtual buses** may be needed.

Analysis tools are developed for particular combinations of **thread** and **connection** protocols and scheduling algorithms. The above data will be common to all, but individual tools may allow or require additional data. ACVIP planners and performers should consider the available tools when making decisions about performing schedulability analysis.

> *Example: Hazard analysis of a model has determined that certain situation awareness capabilities are safety-critical. They have a Design Assurance Level (DAL) sufficient for airworthiness authorities to require analytic verification of timing properties for all software and hardware that implements or affects those capabilities. The derived PDR model provides these capabilities using software and hardware components that are isolated (partitioned) from other components. The derived CDR model uses only periodic threads that are hosted on processors that comply with the ARINC 653 standard and binds connections to a switched Ethernet that complies with the ARINC 664 standard. Schedulability analysis is performed using an ARINC 653 schedulability analysis (static schedule verification) tool obtained from the selected RTOS vendor and an ARINC 664 schedulability analysis (real-time network calculus) tool obtained from the selected network vendor, applied within a compositional schedulability analysis framework[2] to analytically verify end to end latency requirements.*

5.6.3 CDR Risk Management Framework Analysis

Analyses can be performed to verify that security control placement is adequate, controls are non-bypass-able, and controls are tamper-resistant. These analyses do not verify the controls themselves are correctly implemented, they verify the controls exist and are properly deployed in the architecture of the system. Model analysis at this stage yields confidence that the system architecture places required security controls properly to protect the flows.

The model developer should annotate the model at the locations where system components will enforce the required security controls. Model analysis should then confirm that every component that manages a flow also protects that flow with the required security controls.

A component's implementation yields several realizations of that component (e.g., as a software process, as an operating system image, as a software partition, etc.), and model analysis should examine all

[2] A compositional analysis framework allows different analysis tools suited for different subsystems and equipment to be integrated in a way that provides end-to-end system timing analysis.

realizations for enforced controls. Model analysis should determine whether or not it is possible to bypass or tamper with a control by bypassing or tampering with one of these realizations.

6. Assure System Conforms to Models

A presumption throughout early development is that the models specify the to-be-built system with sufficient precision and certainty to significantly reduce rework cost and program risk. ACVIP plans should address how this assurance is provided to the degree necessary to achieve the cost and schedule reduction goals. Models are abstract approximations of the actual system, and plans should address how calibration is performed to assess the degree of conformance between model and as-built system, to an acceptable degree of assurance.

Where modeling and analysis is used to provide evidence for certification authorities, a much higher level of assurance is needed to assure that the as-built system conforms to the model-based evidence than is needed for project cost and risk reduction alone as discussed in Support Certification Approvals and Readiness Reviews.

6.1 Use Models as Specifications

Models are key elements of the requirements and specifications that products must satisfy. The ACVIP Management Plan should include activities to assure that the final as-built components and system comply with the specification models used during early-phase virtual integration and analysis. There should be early-phase tasks to assure that other early-phase work products comply with their model-based specifications. Models should be analyzed for complexity, implementability, etc., to assure that planned processes and technologies are able to dependably produce products that comply with those models.

Validation determines if a requirements model satisfies the needs of the users. Analysis of requirements models performs model validation rather than model verification because this detects inconsistent, incomplete, or unsatisfiable requirements. These are defects in the model-based specifications.

Verification determines if a system complies with its specification model. If a system does not conform to a valid model, then those defects are in the system rather than the model. ACVIP analyses focus on detecting defects in the models. At Certification and Readiness reviews and Physical Configuration Audit, evidence that the as-built system complies with its specification model must be provided.

During early development phases, this is forward-looking and requires management controls during system design, implementation, and integration. Assurance is also required that the final as-built system conforms to a final delivered model to a degree sufficient for the purpose. There are at least two scenarios in which this is required.

- The final delivered models must accurately describe the as-built system for the purpose of streamlining subsequent upgrade projects. Where the Program ACVIP Plan calls for delivery of models to support future upgrades, the ACVIP Management Plan should explain how this will be accomplished during the Physical Configuration Audit (PCA).

- Where models and analyses are used to provide supporting evidence for certification and approval reviews, the ACVIP Management Plan should explain how the necessary degree of conformance between the system and the models and analyses is to be assured at those reviews.

6.2 Generate Implementation Artifacts from Models

Assurance that an as-built system conforms to its model can be increased by automatically generating detailed design and implementation artifacts from the model. Assets that are generated from AADL architecture models are glue code and configuration files rather than code for software application algorithms or hardware circuit designs. The level of assurance can be increased by assuring the generation tools and/or by independently verifying the generated assets using a combination of review, analysis and testing.

> *Example: A mission system integrator is hosting multiple FACE UoPs on an ARINC 653 compute module. Transport Services (TS) functions are implemented using a combination of a configurable software layer in each partition that contains a UoP and the RTOS inter-partition messaging services. The software layer in each partition is configured by modifying some of its code, and a tool is used to generate this code from the AADL model. A second tool is used to generate an AADL schedule from the model. A third tool is used to generate the ARINC 653 RTOS configuration file used to integrate the hosted UoPs and their partitions. A fourth tool is used to automate the make/build process to create a bootable load image.*

Certification credit may be obtained for implementation artifacts that are automatically generated from the analytically-verified model if the generation tool has been sufficiently assured or if an independent verification tool or method is applied to the generated result.

6.3 Perform Model-Based Testing

Assurance that an as-built system conforms to its model can be increased by using model-based testing methods and tools [25]. As with analysis, model-based testing should address requirements, and plans should consider cost versus benefit.

It may be useful to elaborate the system model to create a System Integration Lab (SIL) model that adds information about verification and validation methods to be applied. The SIL model may add components such as emulators and test equipment for environment objects. The SIL model may, with careful consideration, substitute special test components for selected system components that have greater controllability or observability.

A SIL model may add model-based verification methods such as automated test generation or automated checks that observed behaviors comply with the model.

> *Example: A System Integration Lab (SIL) will be used to verify a sample mission system product before it is integrated into an air vehicle. The SIL has a configurable AADL Verification Architecture Model of the configurable lab infrastructure and its suite of simulation, test, user interface, and other lab equipment. The AADL Integration Architecture Model of the sample product is virtually integrated into the AADL Verification Architecture Model of the SIL, and variation point selections*

are made for the equipment and configuration needed for each planned verification task. The Verification Architecture Model includes components to simulate external equipment, such as a Global Positioning System (GPS) emulator. The Verification Architecture Model substitutes for the system network a SIL network that is configurable and instrumented for high-speed collection of message data. The resulting model is used to automatically generate configuration data for some lab equipment (such as configurable network crossbar switches) and used for automated model-based testing to verify the sample system conforms to the SIL's AADL Verification Architecture Model.

Appendix A: ACVIP Management Plan Checklist

- ☐ ACVIP Management Plan is consistent with the System Engineering Management Plan
- ☐ ACVIP risk management goals and plans are consistent with the Risk Management Plan
- ☐ ACVIP certification evidence goals and plans are consistent with the relevant certification plans
- ☐ ACVIP Management Plan identifies the ACVIP goals and requirements for the project
- ☐ Categories of defects and rework targeted for early detection and reduction are identified
- ☐ Models to be developed or reused are identified
- ☐ Models to be delivered from one organization to another are identified
- ☐ The content and structure of delivered models is consistent with model access control plans
- ☐ The purpose, model, and analyses to be performed at each review are identified
- ☐ Plan identifies potential future upgrades to be accommodated by the model
- ☐ Cost versus benefit assessment was done and the rationale is documented
- ☐ Model-based descriptions including needed libraries and patterns will be provided to model suppliers
- ☐ Dependencies between and delivery schedules for models are consistent with project plans
- ☐ Change and configuration management plan is in place and adequate for model exchanges
- ☐ The technical information needed to develop each model will be available when needed
- ☐ Plan identifies which models are the sources-of-truth for key pieces of information
- ☐ Procedures are identified to take corrective and preventative actions after model development
- ☐ Procedures are identified to take corrective and preventative actions after virtual integrations
- ☐ Plan establishes traceability from higher-level requirements to SRR model
- ☐ SRR model establishes patterns and conventions to elaborate traceability through subsequent reviews
- ☐ Plan establishes traceability from CDR AADL model to component non-AADL models and specifications
- ☐ Plan establishes traceability from CDR models through certification and acceptance reviews
- ☐ Plan provides sufficient time and resources to perform virtual integration and analysis activities
- ☐ Plan allows for collaboration with model suppliers during virtual integrations to resolve problems
- ☐ Models are included as part of the specifications for component design and implementation
- ☐ Models are included as part of the specifications for how components are to be integrated
- ☐ Plan provides adequate assurance the as-built system conforms to its specification models

References

[1] Office of the Deputy Assistant Secretary of Defense for Systems Engineering, "Digital Engineering Strategy," Department of Defense, 2018.

[2] Defense Acquisition Notes, "Modular Open Systems Approach," [Online]. Available: http://acqnotes.com/acqnote/careerfields/modular-open-systems-approach. [Accessed July 2020].

[3] AVSI, "System Architecture Virtual Integration," [Online]. Available: https://savi.avsi.aero/. [Accessed March 2021].

[4] J. Hanseso, P. Feiler and S. Helton, "ROI Analysis of the System Architecture Virtual Integration Initiative," Software Engineering Institute, Pittsburgh, PA, 2011.

[5] SEI, "Architecture-Centric Virtual Integration Overview with the Architecture Analysis and Design Language," Software Engineering Institute, Pittsburgh, PA, 2016.

[6] SEI, "Architecture-Centric Virtual Integration Process (ACVIP) Acquisition Management Handbook," Software Engineering Institute, Pittsburgh, PA, 2021.

[7] SAE, "Architecture Analysis and Design Language," SAE International, Warrendale, PA, 2012.

[8] International Council on Systems Engineering, "Model-Based Systems Engineering," Object Management Group, [Online]. Available: http://www.omgwiki.org/MBSE/doku.php. [Accessed 27 August 2015].

[9] DoD, "Risk Management Guide for DoD Acquisition," United States Department of Defense, Washington, DC, 2006.

[10] J. H. Hayes, "Building a Requirement Fault Taxonomy: Experiences from a NASA Verification," in *14th International Symposium on Reliability Engineering*, 2003.

[11] R. R. Lutz, "Analyzing Software Requirements Errors," in *IEEE International Symposium on Requirements Engineering*, 1993.

[12] C. A. M. S. a. K. E.-E. Oliver Laitenberger, "An Experimental Comparison of Reading Techniques for Defect Detection in UML Design Documents," National Research Council Canada, 1999.

[13] Wikipedia, "Sensitivity Analysis," Wikipedia, San Francisco, CA, 2017.

[14] L. M. N. Paul C. Clements, Software Product Lines: Practices and Patterns, Addison-Wesley Professional, 2001.

[15] J. S. B. D. G. M. R. L. J. D. Bill Haskins, "Error Cost Escalation Through the Project Life Cycle," *INCOSE International Symposium,* vol. 14, no. 1, 2014.

[16] P. H. Feiler, J. Hansson, D. de Niz and L. Wrage, "System Architecture Virtual Integration: An Industrial Case Study," Software Engineering Institute, Pittsburgh, PA, 2009.

[17] F. Shull, V. Basili, B. Boehm, A. Brown, P. Costa, M. Lindvall, D. Port, I. Rus, R. Tesoriero and M. Zelkowitz, "What We Have Learned About Fighting Defects," *Proceedings of the 8th Symposium on Software Metrics,* 2002.

[18] D. Ward and S. Helton, "Estimating Return on Investment for SAVI (a Model-Based Virtual Integration Process)," *SAE International Journal of Aerospace,* 2011.

[19] Adventium Labs, "Joint Common Architecture Shadow Architecture Centric Virtual Integration Process Demonstration," 2015.

[20] Object Management Group, "About the OMG System Modeling Language Specification Version 1.6," 2019. [Online]. Available: https://www.omg.org/spec/SysML/About-SysML/. [Accessed 2019].

[21] DoD, "The DoDAF Architecture Framework Version 2.02," DoD, [Online]. Available: http://dodcio.defense.gov/Library/DoD-Architecture-Framework/. [Accessed November 2017].

[22] The Open Group, "FACE Technical Standard, Edition 3.0," 2017.

[23] IEEE, "Recommended Practice for Architecture Description of Software-Intensive Systems," IEEE, 2009.

[24] J. Tretmans, "Test Generation with Inputs, Outputs and Repetitive Quiescence," University of Twente, Endhoven, 1996.

[25] A. C. D. Neto, R. Subramanyan, M. Vieira and G. H. Travassos, "A Survey on Model-based Testing Approaches: A Systematic Review," in *ACM international workshop on Empirical assessment of software engineering languages and technologies*, New York, 2007.

[26] SAE, "SAE Architecture Analysis and Design Language (AADL) Annex Volume 2: Behavior Model Annex (draft AS5506/2)," SAE International, Warrendale, PA, 2016.

[27] FAA, "FAA AR-08/32 Requirements Engineering Handbook," FAA, 2009.

[28] V. P. Nelson, "Fault Tolerant Computing: Fundamental Concepts," *IEEE Computer,* vol. 3, no. 7, 1990.

[29] A. Avizienis, J.-C. Laprie, B. Randell and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Transactions on Dependable and Secure Computing,* vol. 1, no. 1, 2004.

[30] SAE, "SAE Architecture Analysis and Design Language (AADL) Annex Volume 1: Annex E: Error-Model Annex," SAE International, 2015.

[31] DoD, "Acquisition Program Technical Certifications Summary," May 2013. [Online]. Available: https://www.acq.osd.mil/se/docs/Acquisition-Program-Technical-Certifications-Summary.pdf. [Accessed 13 November 2017].

[32] DoD, "MIL-STD-882E Standard Practice for System Safety," Washington, DC, 2012.

[33] SAE, "Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment," 1996.

[34] N. Leveson, Engineering a Safer World: Systems Thinking Applied to Safety, Cambridge, MA: MIT Press, 2011.

[35] N. Leveson, C. Wilkerson, C. Fleming, J. Thomas and I. Nancy, "A Comparison of the STPA and the ARP 4761 Assessment Process," MIT, Cambridge, MA, 2014.

[36] N. G. Leveson and J. P. Thomas, STPA Handbook, 2018.

[37] DoD, "DoDD 5030.61 Airworthiness Policy," DoD, Washington, DC, 2015.

[38] US Army, "Airworthiness Qualification of Aircraft Systems," Headquarters, Department of the Army, Washington, DC, 2007.

[39] CCDevCom, "Systems Readiness Directorate (SRD)," [Online]. Available: https://www.avmc.army.mil/Directorates/SRD/. [Accessed 29 July 2020].

[40] DoD, "DoDI 8540.01 Cross Domain Policy," DoD, Washington, DC, 2017.

[41] DoD, "DoDI 8510.01 Risk Management Framework (RMF) for DoD Information Technology (IT)," US Department of Defense, Washington, DC, 2017.

[42] DoD, "DoDI 8500.01 Cybersecurity," US Department of Defense, Washington, DC, 2014.

[43] NIST, "Framework for Improving Critical Infrastructure Cybersecurity," National Institute of Standards and Technologies, Boulder, CO, 2014.

[44] U. S. Army, "AR 25-2 Army Cybersecurity," 2019.

[45] DoD, "Defense Acquisition Guidebook," [Online]. Available: https://www.dau.edu/tools/dag. [Accessed March 2021].

[46] R. L. Graham, "Bounds on Multiprocessing Timing Anomalies," *SIAM Journal on Applied Mathematics,* vol. 7, no. 1, 1969.

[47] P. Axer and others, "Building timing predictable embedded systems," *ACM Transactions on Embedded Computing Systems,* vol. 13, no. 4, 2014.

[48] S. Kleiner and C. Kramer, Model Based Design with Systems Engineering Based on RFLP Using V6, Berlin: Springer, 2013.

[49] DoD, "DoDI 5000.02 Operation of the Defense Acquisition System," 2017.

[50] D. Redman, D. Ward, J. Chilenski and G. Pollari, "Virtual Integration for Improved System Design," in *Analytic Virtual Integration of Cyber-Physical Systems*, 2010.

List of Acronyms

3D – Three Dimension
ADL – Architecture Description Language
AADL – Architecture Analysis and Design Language
ACVIP – Architecture Centric Virtual Integration Process
AMACC -- Army Military Airworthiness Certification Criteria
AQP – Airworthiness Qualification Plan
AQS – Airworthiness Qualification Specification
AvMC – Aviation & Missile Center
API – Application Programming Interface
ARINC – Aeronautical Radio, Incorporated
ARP – Aerospace Recommended Practice
ASoT – Authoritative Source of Truth
BPS – Bits Per Second
CAD – Computer Aided Design
CAI – Critical Application Items
CCA – Common Cause Analysis
CD – Cross Domain
CDD – Capability Development Document
CDR – Critical Design Review
CDRL – Contract Data Requirements List
CDS – Cross Domain Solution
CIA – Confidentiality, Integrity, Availability
CNSS -- Committee on National Security Systems
CONOPS – Concept of Operations
CSC – Computer Software Component
CSCI – Computer Software Configuration Item
CSI – Critical Safety Item
CSU – Computer Software Unit
DAL – Design Assurance Level
DD – Dependency Diagram
DEVCOM – Development Command
DID – Data Item Description
DoD – Department of Defense
DoDAF – Department of Defense Architecture Framework
DoDI – Department of Defense Instruction
DT&E – Developmental Test and Evaluation
EMD – Engineering & Manufacturing Development
EMV2 – Error Model Version 2

ESOH – Environment, Safety, and Occupational Health
ETA – Event Tree Analysis
FAA – Federal Aviation Administration
FACE™ – Future Airborne Capability Environment
FHA – Functional Hazard Assessment
FMEA – Failure Modes and Effects Analysis
FMECA – Failure Modes, Effects, and Criticality Analysis
FTA – Fault Tree Analysis
GFI – Government Furnished Information
GPR – Government Purpose Rights
GPS – Global Positioning System
I/O – Input/Output
ICD – Interface Control Document
IDE – Integrated Development Environment
IT – Information Technology
JMR – Joint Multi-Role
LCC – Life Cycle Cost
M&S – Modeling and Simulation
MA – Markov Analysis
MIL-HDBK – Military Handbook
MIL-STD – Military Standard
MILS – Multiple Independent Levels of Security
MIPS – Millions of Instructions Per Second
MDA – Model Driven Architecture
MLS – Multiple Levels of Security
MODAF – Ministry Of Defense Architecture Framework
MSAD – Mission System Architecture Demonstrations
ms – Milliseconds
MSI – Mission System Integrator
NASA – National Aeronautics and Space Administration
NIST – National Institute of Standards and Technology
NSS – National Security Systems
OMG – Object Management Group
OSA – Open Systems Architecture
OSATE – Open Source AADL Tool Environment
OSLC – Open Services Lifecycle Collaboration
PCA – Physical Configuration Audit
PHAC – Plan for Hardware Aspects of Certification
PDR – Preliminary Design Review
PSAC – Plan for Software Aspects of Certification
PSSA – Preliminary System Safety Assessment
RAM – Reliability, Availability, Maintainability

RBD – Reliability Block Diagram
RDAL – Requirements Definition and Analysis Language
RDECOM – Research, Development and Engineering Command
ReqIF – Requirements Interchange Format
RMF – Risk Management Framework
RTCA – Radio Technical Commission for Aeronautics
RTOS – Real Time Operating System
S&T – Science and Technology
SA – Situation Awareness
SATCOM – Satellite Communications
SEI – Software Engineering Institute
SEP – Systems Engineering Plan
SEMP – Systems Engineering Management Plan
SINCGARS - Single Channel Ground and Airborne Radio System
SIL – System Integration Lab
SP – Special Publication
SRD – Systems Readiness Directorate
SRR – System Requirements Review
SSA – System Safety Assessment
STPA – System-Theoretic Process Analysis
SysML – System Modeling Language
TD – Technology Demonstrator
TDP – Technical Data Package
TOC – Total Ownership Cost
TRR – Test Readiness Review
TSS – Transport Services Segment
UAV – Unmanned Air Vehicle
UCDSMO – Unified Cross Domain Services Management Office
UDDL – Universal Domain Description Language
UML – Unified Modeling Language
UPDM – Unified Profile for DoDAF/MODAF
UoP – Unit of Portability
USM – UoP Supplied Model
VHDL – VHSIC Hardware Description Language
VHSIC – Very High Speed Integrated Circuit