SCIENTIFIC TEST
& ANALYSIS
TECHNIQUES
CENTER OF EXCELLENCE

Model Based Systems Engineering for Developmental System Verification and Validation

August 2022

Chuck Rogal, CTR

The STAT COE provides independent STAT consultation to designated acquisition programs and special projects to improve Test & Evaluation (T&E) rigor, effectiveness, & efficiency.

About this Publication:
This work was conducted by the Scientific Test & Analysis Techniques Center of Excellence under contract FA8075-18-D-0002, Task FA8075-21-F-0074.

For more information:
Visit, www.AFIT.edu/STAT
Email, CoE@afit.edu
Call, 937-255-3636 x4736

Scientific Test & Analysis Techniques Center of Excellence
2950 Hobson Way
Wright-Patterson Air Force Base, Ohio

Version: 1, FY22

Modernizing *the* Culture *of* Test & Evaluation

Executive Summary

Model-Based Systems Engineering (MBSE) can be applied to various types of system building. MBSE can also highlight security considerations across a developing architecture. This paper illustrates a view of MBSE for software-driven systems which can be subsequently mapped to the traditional V-model. This approach shows a validation-based methodology linked to a verification-driven progression for decomposing and integrating a system in a platform independent way. This MBSE Platform Independent Model (PIM) approach can be the anchor of a Shift-Left change for the sake of cost and schedule. Additionally, this approach emphasizes risk-reduction, artifact reuse, and simplicity.

Keywords: model-based systems engineering (MBSE), systems engineering (SE), validation, verification

Table of Contents

Introduction

The traditional systems engineering (SE) process involves decomposing a system to produce a set of hardware and software specifications. This SE process typically excludes validation and verification (V&V) planning up front, and often results in discovering risk too late. Software development specifications often lack detail, increasing risk and may result in a system that does not meet the operational need. Specifically, the specifications do not include validation and verification (V&V) planning and omit critical information such as timing and sequence expectations. Additionally, requirements often fail to identify runtime expectations and required conditions. Without this information, implementers must make uncomfortable decisions regarding what is supposed to happen when including algorithm instantiation and results. This paper presents a model-based systems engineering (MBSE) approach that uses a Platform Independent Model (PIM) to improve current MBSE and SE practices. The MBSE PIM promotes a V&V-driven system decomposition and offers a new methodology for test and evaluation (T&E) individuals to consider implementing. The MBSE PIM provides opportunities for early testing, an architecture for deriving security requirements, and a view of capabilities which is otherwise unavailable. The MBSE PIM promotes artifact re-use, aligns with the Shift-Left mentality by providing V&V participation much earlier in a program, identifies risk sooner, and expedites mitigation actions.

The remainder of this paper includes a background section discussing the traditional SE process, role of MBSE, and the intended audience. Then, the paper provides an overview of the MBSE PIM followed by a detailed discussion of the PIM Hierarchy. The PIM Hierarchy consists of 4 phases. The first phase, Concept of Operations (CONOPS) PIM Pass 1 includes: capturing user intent (e.g., Use Cases, Data Dictionary, Key Performance Parameters (KPPs), and Scenarios), and identifying referents (i.e., authoritative sources). The second phase, CONOPS PIM Pass 2 builds upon the first phase by characterizing data in the Data Dictionary, creating a draft allocation, developing Sequence Diagrams, creating Technical Performance Measures (TPMs), refining Scenarios, and developing security requirements. Pass 2 transforms the notional model into a top-level draft architecture. The activities of the third phase, Top-Level Discrete Event PIM, are refining the model (e.g., external input/output characterization, algorithm locations, and security requirements) and building the executable model (which requires an actual modeling tool). The final phase is the Lower-Level Discrete Event PIM which creates a design solution with sufficient detail to be implemented and verified. The paper concludes with a summary of key takeaways.

Background

The Department of Defense (DOD) Systems Engineering Guidebook (2022) defines systems engineering as “the primary means for determining whether and how the challenge proposed by a program’s requirements can be met with available resources” (p. 5). DOD SE practices consist of 16 distinct processes, split between technical and technical management processes (see Figure 1).

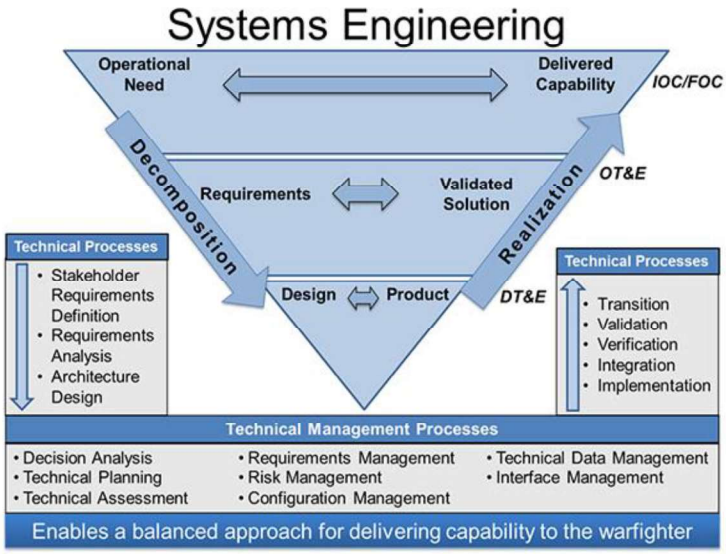

Figure 1
DOD Systems Engineering Process

Note. Figure from Systems Engineering Guidebook (2022). Department of Defense. Copyright 2022 by Department of Defense.

A more traditional SE approach is shown in Figure 2 and is referred to as the V-Model.

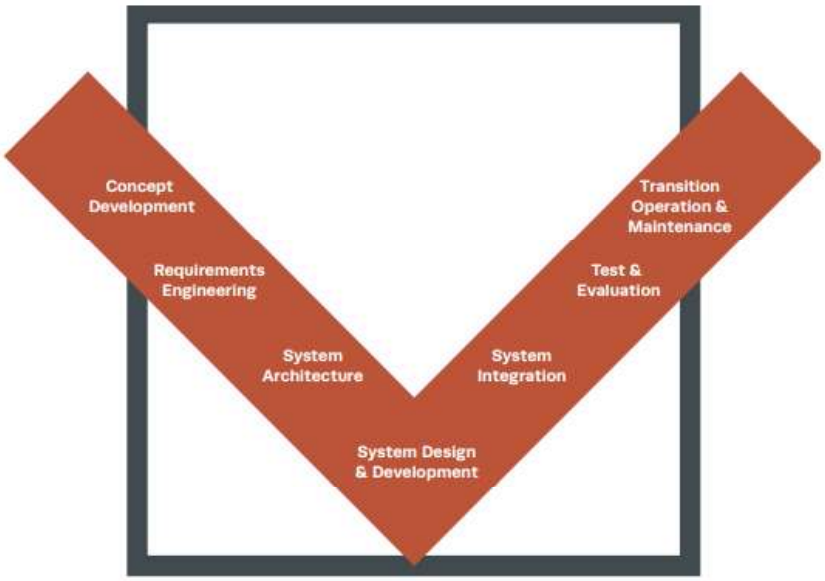

Figure 2
V-Model of Systems Engineering Lifecycle

Note. Adapted from MITRE Systems Engineering Guide (2014). The MITRE Corporation. Copyright 2014 by the MITRE Corporation.

When using this V-Model approach, division between the front-end work (i.e., left-side of the V-Model) and the back end (i.e., right-side of the V-Model) results in risk becoming evident mostly when it is too late to mitigate. The back end is at best weakly connected to the front part of the

V-Model activity, resulting in incomplete systems with missing information (i.e., timing, sequence, and V&V planning).

This paper defines MBSE as a hierarchical set of models meant to address the front-end work for the software-driven portions of some application. MBSE can identify risk earlier (i.e., the front of the process) by explicitly connecting the front end to the back end of the effort. MBSE creates this connection by converting much of test planning to a front-end activity. Utilizing MBSE can cement both sides of the job together and promote artifact reuse. Specifically, this paper offers a PIM approach which does not include any implementation choices but can include an allocation to standardized components. Thus, this approach is broadly applicable since it does not require a specific modeling tool.

The intended audiences for this paper are T&E practitioners historically on the back end, and program managers and systems engineers on the front end of the V-Model. The goal is that all stakeholders adopt this methodology and the T&E practitioners and program managers become more involved with the front end to identify risk sooner. A list of acronyms and definitions is provided in Appendix B to aid in understanding of many MBSE terms.

MBSE PIM Overview

The Model-Based Systems Engineering Platform Independent Model (MBSE PIM) offers validation opportunities during each phase of development and integrates directly with the traditional V-Model. Furthermore, validation is much easier with the benefit of a model since the model provides a better reference for the system than what appears in typical documentation. Figure 3 shows how the MBSE PIM integrates into the traditional V-Model and additionally provides earlier opportunities for T&E.

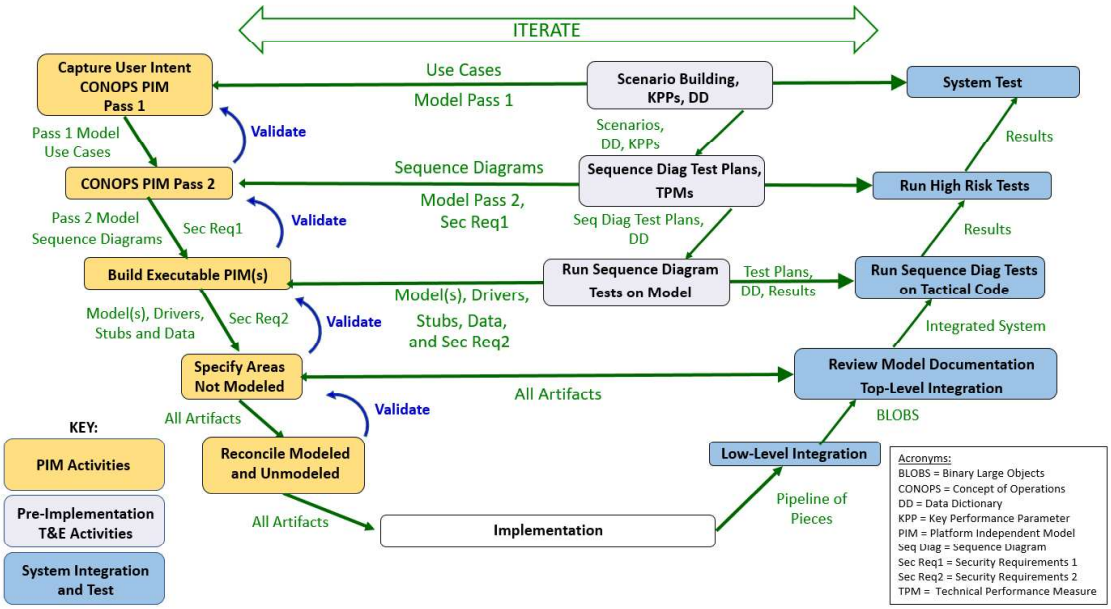

Figure 3
MBSE PIM Mapped to V-Model

Figure 3 maps the MBSE PIM to the left side of the V-Model and identifies artifacts which are passed between activities with green arrows. Artifacts are passed down as the PIM increases in granularity, across to the left side to connect T&E activities to developmental activities, and up

the right side of the V-Model as the system is integrated and tested. The left and right sides are connected through iterative communication and passing of artifacts to ensure user needs will be met.

The MBSE PIM facilitates validation on the left side of the V-Model, which is then checked as the right side is traversed upwards. The validation should be checked on the right side in **Error! Reference source not found.** to revisit the validation completed on the left side and ensure everything is correct and nothing was overlooked. Once the last validation check is finished (i.e., the top right side of the V-model), validation is complete.

In the pre-modeling phase, initial validation for a novel application of technology must be based on referents derived from similar legacy systems since the application has no true precedent. The validation may be somewhat qualitative since, at this point, the notional system exists only as a collection of CONOPS relevant information. This is the period where the CONOPS information is initially validated and evaluated for model building. Referents for this pre-modeling phase are listed below:

- user inputs
- similar systems successfully fielded
    - evaluate inputs and test results from such systems
    - collect, analyze, and review potential input sources and include timing
    - evaluate output destinations including timing and document the same
- trade studies of risk vs technology maturity to arrive at the best mix of both
- related models and simulations (if they exist)
- where the system would fit within the defense structure (tactical/strategic/theater)
- timing studies to affirm system posture within the defense structure
- subject matter expert (SME) inputs

In addition to identifying referents, modeling tools must be evaluated. The appropriate modeling tool would offer a mature grammar, open interfaces, discrete math and continuous math, graphical representations, a simulation engine, and configurable clock objects. Ideally, the modeling tool would be available from the beginning.

The first two levels of the model, as shown in Figure 3, can be worked on informally prior to modeling tool selection. This modeling approach includes the following assumptions:

- Data relationships within the application do not require a separate data model. The model as defined in this paper will expose those data relationships in the context of the modeling layers and the associated operations
- Application will resolve to a mix of discrete (integral) and continuous (floating-point operations)
- Models will offer up new verification opportunities. These opportunities will be leveraged to produce more coverage and a chance to tune the accuracy, precision, and acceptable range of inputs and outputs (I/O)
- Application is built with proven and trusted technology for safety and reliability, maintainability, and availability (RAM). Such technologies should have been previously used successfully in the field and can consist of both hardware and software
- Iteration is part of the approach regardless of whether an agile, waterfall, or spiral methodology is used to build the system

Next, the paper discusses PIM Hierarchy and includes detailed descriptions of each phase.

PIM Hierarchy

The PIM Hierarchy identifies validation opportunities and layered verification locations. The progression which follows will highlight the front-end part of a PIM. PIMs steer clear of the challenging decisions to be made on a Platform Specific Model (PSM) path. PIMs also offer the content needed to move on to an implementation whether automated or not. This focus is due to that fact that building a PIM nicely illustrates the changes needed for better T&E. The PIM Hierarchy consists of the following four phases:

- CONOPS PIM Pass 1
- CONOPS PIM Pass 2
- Top-Level Discrete Event PIM
- Lower-Level Discrete Event PIM

Figure 4 shows the PIM Hierarchy decomposed into the four phases. Each phase includes some lower-level details for clarification. Once the final phase is complete, the full model is implemented.

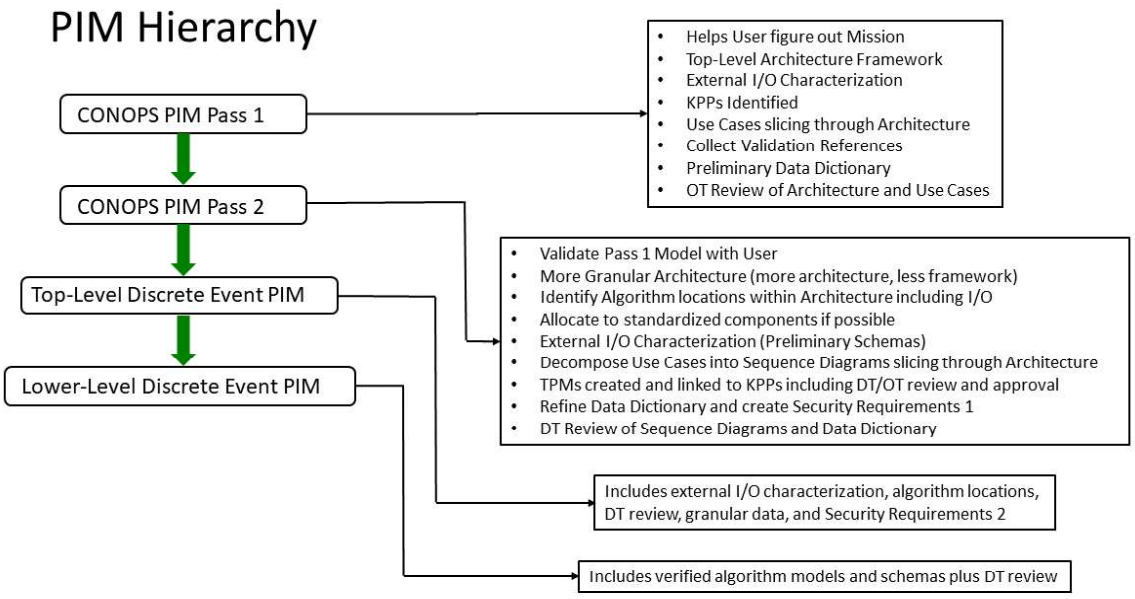

Figure 4
Front-End PIM Progression

CONOPS PIM Pass 1

The first phase of the PIM Hierarchy is CONOPS PIM Pass 1. This phase is a notional representation. At the top-level, there is only a single model representation of the system. Additional models will most likely be developed at lower layers of the modeling hierarchy. Many applications will include lower layers with stochastic elements in the form of algorithms. There may be algorithm models to represent the stochastic parts of the system. The algorithms may digest floating-point and integral inputs and produce both as outputs.

CONOPS validation should show the system is aligned with the needs of the warfighter and assure the model contains the correct application. A set of Use Cases will be developed to accompany the model. All Use Cases will start at the model context boundaries (i.e., external interfaces). The strength of the Use Cases lies in bounding the mission in time and creating

start and end points for Scenarios. The Use Cases will contain no allocation of functions to components and are therefore logical representations of sequences. The Use Case sequences in Pass 1 will evolve into Sequence Diagrams in Pass 2 once the initial allocation is complete. Mission requirements can be derived from the Use Cases and other prepared documentation. The artifacts developed during the CONOPS PIM Pass 1 Phase are:

- Notional Model
- Use Cases
- Top-Level Data Dictionary (DD)
- Key Performance Parameters (KPPs)
- Scenarios
- Mission Requirements derived from the other artifacts

The model is partitioned functionality within a system perimeter in a graphical representation. Essentially, the system is bounded by a perimeter (i.e., external interfaces) with functionality spread over a set of elements (i.e., partitioned functionality) and the whole system is represented graphically. The Use Cases should traverse pieces of functionality within the model, exercising all functions and all major logical paths. The Scenarios can serve to drive the Use Cases. The Data Dictionary is used to build the Scenarios. The Data Dictionary fields are shown in Table 1 and provide vital information for downstream stakeholders. The fields are populated during the entire front-end of the V-Model.

Table 1
Data Dictionary Fields

Name	Description	Source Interface	Destination Interface	Synch/ Asynch	Frequency	Type	Size	Acceptable Range	Units	Precision

Table 2 provides descriptions of the fields in Table 1.

Table 2
Data Dictionary Field Descriptions

Data Item Name	Data Item Description
Name	Name of data item
Description	Description of data item
Source Interface	Name of external interface when applicable
Destination Interface	Name of external interface when applicable
Synch/Asynch	Communication mode when applicable
Frequency	Only for synchronous communication
Type	Data type initially only for external interface data
Size	Size initially only for external interface data
Acceptable Range	Initially only for external interface data
Units	Measured quantities in some familiar system (e.g., MKS)
Precision	Significant digits

The KPPs provide a sense of expected performance across the model and are built into the requirements. The Use Cases can be thought of as referent extensions since the referents capture user need and CONOPS, and should serve to drive the model in the right direction. The

mission requirements are derived from the other artifacts (e.g., Use Cases, Data Dictionary, etc.) and will be illustrated by the model hierarchy.

In the spirit of "are we building the right model," validation will consider the artifacts above and be anchored by the information gathered in the pre-modeling phase. In other words, validation during this phase ensures the Pass 1 model and developed artifacts meet the user needs and CONOPS. Additionally, the activities below will strengthen the validation activity:

- Review the model, Use Cases, and requirements including KPPs,
- Review first pass of the Hierarchical Data Dictionary, and
- Conduct dry-run Scenarios based on the above artifacts.
- Operational Test (OT) personnel and users review all artifacts. This may be a new role for the OT personnel but ensures their involvement with the CONOPS. The benefits of their involvement will pay dividends throughout the rest of the development cycle.

Validation results will be provided to the model engineers and other involved stakeholders. Assuming a green light is given for the results of this phase, then the work moves on to the CONOPS PIM Pass 2 Phase.

CONOPS PIM Pass 2

The CONOPS PIM Pass 2 phase refines the notational model developed in phase 1 and creates a functional allocation of the system for each part of the model. Each part needs to be defined and include decisions about hardware versus software allocations. Weighing heavily on these decisions are the on-going standardization efforts of Government and Industry (e.g., FACE, SOSA, OMS, VITA). A draft physical architecture is created from the logical architecture developed in CONOPS PIM Pass 1. The developing physical architecture is still very much in the realm of a PIM since the architecture components are standardized and available across platforms. The CONOPS PIM Pass 2 draft architecture can also be thought of as a reference architecture for the system under development. Pass 2 of the CONOPS Model includes the following activities:

- Transform notional model into top-level draft architecture
- Transform Use Cases into Sequence Diagrams with draft allocations to hardware and software
- Capture performance considerations (e.g., throughput, network utilization, size, weight and power (SWAP), reliability, availability, maintainability (RAM), etc.)
- Create TPMs and link them to KPPs
- Create top-level security requirements
- Extend/refine Data Dictionary (populate more columns)
- Extend/refine requirements (identify verification data)
- Extend/refine and document Scenarios (stitched together better)

Performance must be considered as an integral part of functionality during this and other phases and as such will be considered during allocation decisions. Separately documenting performance requirements is not recommended. When separated, the implementers are faced with reconciling the separation while building the system. Performance should be an inseparable part of functionality. With that thought in mind, TPMs will be created and linked upward to the KPPs of the previous phase. Additional requirements will be derived organically from the modeling artifacts. MBSE changes the nature of requirements since the model will illustrate the expected functionality which makes the requirements a test reference only. Therefore, the requirements are relieved from capturing the story of the system since that functionality is exposed by the model.

The Sequence Diagram work can be performed in lockstep with the allocation work. The Sequence Diagrams will be driven by the Scenarios and will illustrate the allocation. A measure of fidelity can be captured in the Acceptable Range, Units, and Precision fields within the Data Dictionary. Recall the Data Dictionary will capture model outputs as well as inputs. When appropriate, the Data Dictionary should also capture algorithm input/output (I/O) in breakout sections. Developmental Test (DT) personnel begin to work on the Sequence Diagram Test Plans as soon as the Sequence Diagrams are delivered across the V-Model. Also, TPMs can be derived from the KPPs and noted as metrics within the Sequence Diagram Test Plans. Top-level security requirements during this pass will involve identifying locations within the developing draft architecture where security considerations are the most pronounced. Other requirements will be derived from the information presented above.

DT personnel perform CONOPS PIM Pass 2 validation to ensure new artifacts meet the needs captured during the previous phase. Specifically, they:

- Use referents to confirm Use Case Translation to Sequence Diagrams
- Confirm associated allocation to hardware and software
- Review performance associated with the allocations
- Review Scenarios with an eye toward data sources
- Review Data Dictionary with an eye toward schema development
- Review requirements against model and other artifacts
- Populate/review DD Acceptable Range column as a validation exercise (Parser can throw out bad payloads at runtime, a major security consideration)
- Populate the Accuracy and Precision columns of the DD
- Review top-level security requirements

Upon successful validation of this phase, transition to the next phase Top-Level Discrete Event PIM.

Top-Level Discrete Event PIM

The Top-Level Discrete Event PIM phase occurs next and includes detailed logic within the allocation. This modeling level is where a modeling tool becomes necessary if the model is to be developed and executed. This is the phase where programming becomes essential. The activities of this phase are listed below:

- Develop more external I/O characterization
- Create sources and sinks for model I/O
- Refine model structure to be more hierarchical
- Bind operations and data to a fixed grammar in graphical representations
- Expose algorithm locations and characterized I/O
- Logically couple discrete data (integral operands) with continuous data (floating-point operands)
- Model outputs characterized in the Data Dictionary
- Refine security requirements
- Identify test points and associate with requirements

The allocations which began in the previous phase will continue here to create a granular hardware/software physical architecture. The security requirements will be refined per interface and will tie into the schemas of the Data Dictionary. Upon the completion of this phase, some artifacts may be passed on to the implementers for building the tactical code. Also, the model(s) will be passed across the V-Model for Sequence Diagram Test execution. Algorithm models will

be documented and have characterized inputs and outputs. The algorithms should arrive from the algorithm developers with scenarios and documented test cases and results. Working the algorithms into the PIM architecture will be a group effort with principal participation by the model builders and algorithm developers. The goal of this phase is to keep the model simple for maximum use by the most stakeholders. Requirements will continue to be derived from the modeling artifacts. Test points will be identified across the developing architecture along with associated verification data.

The model at this level will be a mix of discrete and continuous operations. Inputs will be simulated. Discrete external interface data will have been parsed and validated. Simulated floating-point telemetry inputs will be validated at the context boundary. Validation of external inputs will consist of parsing when possible, type checking, and range checking. All external data will be sourced from the Data Dictionary. Model operations will be somewhat grammar dependent. The modeling tool should offer the full range of integral and floating-point operations plus logical constructs.

Modeling structures will most likely be a hierarchical set of abstract-communicating finite-state machines (FSMs). The FSM hierarchy will elaborate the application design down to the level necessary for communicating the design to the implementers or for PSM creation. One of the keys to the hierarchy will be to align the stochastic algorithm outputs with the discrete undercarriage to form a set of expected results when executed. The models can be virtualized which will help to execute additional test cases for increased coverage. Traceability will also be somewhat self-evident given the model hierarchy. Executable model hierarchy work will be split between this phase and the next. This phase may bring the model hierarchy to perhaps uneven levels. The next Lower-Level Discrete Event PIM phase will create a balanced model hierarchy at a consistent level. As the model develops, it is important to maintain interface integrity across the model architecture. Interface integrity applies up and down as well as laterally across the architecture.

Model validation during this pass will have several orientations:

- Built-in via type checking and range checking of I/O, operands, and results (can be extended to include intermediate results)
- Review of validation chain up through modeling levels to minimally include referents and user intentions
- Design reviews for completeness, coverage, and implementation readiness
- Review of Sequence Diagram Test Plans and associated results
- Review of lower-level tests and results
- Review of requirements hierarchy reflecting verification levels

Once the model is validated, progress to the final phase to create a design solution.

Lower-Level Discrete Event PIM

This lower level of modeling work completes the PIM Hierarchy. The main idea of this level is to create a design solution with sufficient detail to be implemented and verified. The day-to-day work is the same as the previous phase. The approach to validation is also the same as the previous phase. This phase is used to level set the architecture of the model. All entities and interfaces will be at the same hierarchical level when complete. Modeling artifacts are then passed to the implementers (and/or PSM developers) and across the V-Model to the DT personnel. The PIM will merge the deterministic and stochastic elements of the design to produce a set of expected results when executed. As the right side of the V-Model is traversed

upward, the validation points discussed herein are revisited for confirmation.

Conclusion

Traditional SE processes typically produce incomplete system (hardware and software) specifications and identify risk too late. Applying MBSE with a PIM, as described herein, can produce more mature systems. Maturity can be a measure of test coverage, security, and critical capabilities. This MBSE PIM approach delivers many engineering advantages and provides the following capabilities: testing early, deriving security requirements, gaining insights into the system otherwise unavailable and promoting artifact re-use. Building a model aligns with the Shift-Left mentality by providing V&V participation much earlier in a program, identifies risk sooner, and expedites mitigation actions.

References

Department of Defense (DOD). 2022. Systems Engineering Guidebook. https://ac.cto.mil/wp-content/uploads/2022/02/Systems-Eng-Guidebook_Feb2022-Cleared-slp.pdf

The MITRE Corporation. (2014). MITRE Systems Engineering Guide, 2. https://www.mitre.org/publications/systems-engineering-guide/systems-engineering-guide/the-evolution-of-systems

Appendix A
Related Reading

Boleng, J., & Procter, S. (2018, May). *What Would Convince DOD Program Managers to use Model-based System Engineering?* [Video]. Retrieved from https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=552923

Hazel, Alex & Towers, James. (2020, November). Good Practice in MBSE Model Verification and Validation. Retrieved from https://www.researchgate.net/publication/347436977_Good_Practice_in_MBSE_Model_Verification_and_Validation

Shevchenko, N., (2020, December 21). An Introduction to Model-Based Systems Engineering (MBSE) [Blog post]. Retrieved from https://insights.sei.cmu.edu/blog/introduction-model-based-systems-engineering-mbse/

Shevchenko, N., (2021, February 22). Requirements in Model-Based Systems Engineering (MBSE) [Blog post]. Retrieved from https://insights.sei.cmu.edu/blog/requirements-in-model-based-systems-engineering-mbse/

Appendix B
Acronyms and Definitions

Acronyms

BLOBS **Binary Large Objects**
A short order way to refer to data as separable entities usually captured in files of various types (txt, bin, dat, log, jpg, etc.)
https://en.wikipedia.org/wiki/Binary_large_object

CONOPS **Concept of Operations**
A CONOPS is one-way systems get born. A CONOPS is a view of a system from both external sources and internal sources. The external sources typically come first in the development sequence and in the production of artifacts. Internal sources usually need an architecture which can develop coincidentally with the internal view of the system. Both views are user directed and move from being informal to formal over time. A CONOPS should be validated based on some set of referents.

FACE **Future Airborne Capability Environment**
Defines a reference architecture to support software portability.
https://en.wikipedia.org/wiki/Future_Airborne_Capability_Environment

INCOSE **International Council on Systems Engineering**
A group which advocates a view of how to build systems.
https://en.wikipedia.org/wiki/International_Council_on_Systems_Engineering

KPP **Key Performance Parameter**
A quantifiable item of interest derived from requirements which will provide a measure of system-wide performance. KPPs should fall out from key requirements. For example, a developing system shall be designed to track 25 simultaneous targets. The value 25 may be a KPP since it can provide a measure of overall system performance. When KPPs are more granular than the previous example they are often referred to as Key System Attributes (KSAs). A KSA example might be defined with respect to a system which processes telemetry from some set of sensors. The system might also have to refresh an operator's screen at some associated rate. The associated requirement could be stated as: The operator console update rate shall be at least 1Hz. The value 1Hz might be the associated KSA.
https://www.dau.edu/acquipedia/pages/ArticleContent.aspx?itemid=203

MKS **Meter, Kilogram, Seconds**
A system of units commonly used in engineering and science.
https://en.wikipedia.org/wiki/MKS_system_of_units

OMG **Object Management Group**
A standards body which adopts and subsequently promotes software engineering standards.
https://www.omg.org/

OMS **Open Mission Systems**

Goal is to develop industry consensus for a non-proprietary mission system architectural standard that enables affordable technical refresh and insertion, simplified mission systems integration, service reuse and interoperability, and competition across the life-cycle.
https://www.vdl.afrl.af.mil/programs/oam/oms.php

PIM **Platform Independent Model**
A model that does not include any implementation choices. PIMs can include an allocation to standardized components which are available across any number of implementations. For example, SOSA oriented elements and FACE conformant software products are two standardized types of components. Ideally a PIM is execution-enabled. PIMs are typically illustrated with a logical architecture which can be used as a V&V reference for the model and for the tactical system.
https://cmu-sei.github.io/DevSecOps-Model/

PSM **Platform Specific Model**
A model that includes implementation choices. PSMs can be derived from PIMs or built from scratch with no PIM reference. PSMs illustrate platform choices throughout a physical architecture. They can guide an implementation across the entire development lifecycle. PSMs are typically the result of many challenging implementation decisions. Where a PSM stops is always a many-sided question.
https://www.omgwiki.org/dido/doku.php?id=dido:public:ra:xapend:xapend.a_glossary:p:psm

RAM **Reliability, Availability, Maintainability**
Three areas of non-functional requirements sometimes referred to as the "Illities." RAM considerations are important during physical architecture development and associated allocation. They can also play into software resilience.
https://www.mitre.org/publications/systems-engineering-guide/acquisition-systems-engineering/integrated-logistics-support/reliability-availability-and-maintainability

SOSA **Sensor Open Systems Architecture**
Created to develop a series of modular open systems architecture specifications for a variety of Multi-INT C4ISR platforms that crosses services and platform environments.
https://militaryembedded.com/comms/communications/sensor-open-systems-architecture-sosa-an-overview

SWAP **Size Weight and Power**
Three important considerations when specifying a new component
https://www.analog.com/en/technical-articles/swap-the-rf-solution.html

TPM **Technical Performance Measure**
A quantifiable item of interest derived from requirements which can be tracked throughout the development cycle. TPMs are a type of metric oriented toward risk, and cost and schedule. Within MBSE they can be created based on the model and then applied. They can be applied again to the real tactical system.
https://acqnotes.com/acqnote/careerfields/technical-performance-measurement

V&V **Verification and Validation**

V&V unfortunately joins two terms with very distinct and separable meanings. Sequentially, Validation comes before Verification when the reference is some system development sequence. Validation is all about building a system that meets the needs of the user. There are many ways to achieve Validation. Validation needs a set of good references (or referents) to be successful. Validation should begin in the beginning (see CONOPS) and be pursued across all phases of development. Verification happens cooperatively but also independently as soon as development provides something to verify. The something can be some part of a model and/or some part of a software architecture. The most granular reference for verification is typically a developer who built some piece of code which is large enough to undergo verification. Verification often starts with a Unit which is just a piece of separable code. A Unit might be an Object in and Object-Oriented implementation. Verification proceeds up through levels (i.e., layers) by necessity since the number of possible test paths is too enormous using any other approach. Early in the progression, the reference for the verification usually becomes "shall" statements in a specification. Specifications are layered just like the associated verification. Verification culminates with Operational Testing which looks at a system from the CONOPS view and consequently verifies operational requirements.
https://www.mitre.org/publications/systems-engineering-guide/se-lifecycle-building-blocks/test-and-evaluation/verification-and-validation#:~:text=Definition%3A%20Verification%20is%20the%20process,the%20most%20realistic%20environment%20achievable

VITA **Versamodule Europe bus (VMEbus) International Trade Association (VITA)**
Promotes the concept of open technology for critical embedded computing as embodied in the many standards.
https://www.vita.com

Definitions

Acceptable Range
Acceptable Range is an important part of data characterization. Data characterization is supposed to happen during system decomposition and should appear as a field in the Data Dictionary. Acceptable Range should be applied to each piece of data particularly those that traverse external interfaces. The Acceptable Range aspect of data characterization also reduces the stochastic influence of external inputs. Acceptable Range also play into schema development and enforcement at runtime. Most interface schemas include an Acceptable Range per datum which can be then enforced by the parser at runtime. This enforcement includes a security dimension since Acceptable Range can also deny malicious payloads designed by nefarious actors attempting to invade a system via its external interfaces.

Accuracy
Accuracy is a measure of the error associated with a datum. Accuracy is another part of data characterization developed during system decomposition. Like Acceptable Range, accuracy is vital for data traversing external interfaces. The expected Accuracy of a datum should be reflected in the Acceptable Range documented in the Data Dictionary. A schema can enforce Accuracy by way of the Acceptable Range.

Agile

Agile is an approach to developing systems which uses bite-sized chunks of functionality as its most obvious distinction from more traditional system development methodologies. The bite-sized chunks are developed in Sprints which are short windows of time in which the bite-sized chunks mature into parts of verified system functionality. The developed pieces will then be integrated into the system. The system may be reduced to a Minimum Viable Product (MVP) which is just a smaller version of the anticipated system. The MVP provides an earlier look at the system than may otherwise be available with other development methodologies.

Algorithm
An Algorithm is a recipe for solving a problem. Algorithms range from a simple step by step sequence in a cookbook like fashion to a wildly complicated set of expressions which lead to a solution of a complex problem.

Allocation
Allocation is an activity which occurs during system decomposition. Allocation involves assigning functionality to elements of hardware and software and may appear hierarchical.

Artifact
An Artifact is an input or a result of system development activity. Artifacts usually appear in sets and can range from binary executables to various types of documentation and data.

CONOPS Model
A CONOPS Model is a representation of a proposed system, ranging from a simple drawing to an executable model. CONOPS Models are usually developed early in the sequence of system development activities. They should be validated by way of a set of referents prior to proceeding with further development.

Data
Data is a type of artifact which comes in many forms. The term Data is the plural form of Datum. Computer data can often resolve to operands during runtime which are manipulated and transformed by the operations embedded in the software.

Data Dictionary
A Data Dictionary is a system development artifact used to capture internal and external data produced and used by a developing system. Data Dictionaries are often hierarchical and are the basis for Schemas and Scenarios.

Data Model
A Data Model shows relationships among pieces of data. They are typically represented with entity-relationship diagrams and are often associated with databases.

Data Relationship
A Data Relationship shows how one piece of data is associated with another. Data relationships are often represented by Data Models and are a static view of how a data architecture might look at a certain time. Data relationships are usually dynamic over the course of an executing application or model. Data models by themselves do not expose the dynamic nature of Data Relationships. The dynamic nature of Data Relationships can be viewed within the executable models developed within an MBSE methodology.

Discrete Event Model

A Discrete Event Model or Simulation is one driven by external stimuli which happen to be discrete as opposed to continuous.
https://en.wikipedia.org/wiki/Discrete-event_simulation

Finite State Machine (FSM)
An FSM is a representation of states connected by edges. They can range from very simple to very complex. FSMs are part of the foundation of computer science.
https://en.wikipedia.org/wiki/Finite-state_machine

Floating-Point Operand
A Floating-Point Operand represents a data type which contains a decimal point. The decimal point can float across the value to represent a wide range of numbers.

Integral Operand
An Integral Operand is integer based and so has no decimal point.

Interface / External Interface
An Interface connects two or more parts of a system. Some Interfaces sit on the boundary of a system and are thus referred to as External Interfaces. Other Interfaces are internal to a system and are the result of system decomposition and connect the various parts.

Model
Models have many definitions and come in many forms. For our purposes a model is a representation of a system. Model development can go through various stages to create higher fidelity versions of the system represented. Models can guide subsequent development in terms of validation, requirements, implementation, and verification. Models can also be loosely referred to as simulations.

Model Based Systems Engineering (MBSE)
MBSE is a model-driven development sequence for deploying and fielding new systems. The model begins at the CONOPS level and then gets decomposed into executable components. In so doing the model defines a V&V architecture.

Model Hierarchy
A Model can be built by way of iterations. Each iteration can offer up the same model with additional layers of functionality. The layers can form a hierarchy and add granularity to the model in terms of operations, data, and interfaces.

Model Operation
Model Operation is another way of saying model execution. Modeling efforts for our purposes should work toward models which execute or operate.

Modeling Tools
Modeling Tools can range from high order languages like c to vendor provided applications which include grammars and structures to make building models easier.

Notional Model
A Notional Model is typically an informal representation. Notional Models can be created with paper and pencil. They often form the basis for more formal representations

Parser

A Parser is typically a software application used to process data. Parsers are built into many other applications. For example, compilers and interpreters all contain a Parser. The data Parsers ingest can take several forms. The data can be fed to the Parser over an interface or the Parser can read the data from a file. There are also other ways to provide data to a Parser. For example, a GUI might need to parse data provided by a user in some field of a window.

Payload
The Payload is usually the data carried by a message within a message-driven system architecture. Many messages are partitioned across a set of protocol headers.
Most of the headers will carry along a payload for the peer protocol to ingest and process.

Physical Architecture
The Physical Architecture is another way of referring to the implementation and ultimately includes both hardware and software.

Platform
Usually refers to a Computing Platform. In that sense, a platform is a short-order way to address either an implementation or the host of an implementation or both. Simply, platform is where the software lives while executing.
https://en.wikipedia.org/wiki/Computing_platform

Precision
Precision defines the number of significant digits to the right of a decimal point within a floating-point value. Precision is an important part of the Data Dictionary.

Referent
A Referent is another way of referring to a reference for something else. The “something else” can be a model, a system, or some other element. Referents are used during validation exercises to increase confidence in the direction of the activity.

Scenario
A Scenario is a set of data which can be used to drive an application. The application can be a model or a tactical system. Scenarios are often associated with test cases. A reasonably sized system can be driven by hundreds of Scenarios.

Schema
A Schema is a formal description of some set of data. Schemas are read and enforced by Parsers. They are a way to capture the rules for manipulating a particular set of data. The data can take many forms.

Sequence Diagram
A Sequence Diagram is a set of operations to be accomplished in a prescribed order. Sequence Diagrams can be built at many levels within an architecture.

Simulation
See Model

Spiral Methodology
A Spiral Methodology was popular in software development circles in the early 2000s. Spiral was an early form of iteration but the increments were larger than in an agile methodology.

Stub
A Stub is a placeholder usually for an interface. A stub identifies a location usually within a software architecture for some future work and serves as a placeholder for the expected effort to build the interface.

Use Case
A Use Case is a representation used to capture a sequence of activities. Use Cases similar to Sequence Diagrams can occurs across a wide range of decomposition levels. They typically appear in a set and can represent the universe of processing expected to be performed by some model or system.

Validation
See V&V

Verification
See V&V

Waterfall Methodology
A Waterfall Methodology was the way systems were developed in the 20th century. Development activities spilled from one to the next in a sequential waterfall fashion.