33rd Annual INCOSE international symposium

hybrid event

Honolulu, HI, USA
July 15 - 20, 2023

Applying MBSE in Space Based Systems Development

Chris Madariaga, Ahmad Bashir, Chris Swickline

Introduction

- Formal application of MBSE to generate system architecture models to support DoD program

- Alignment with current DoD Digital Engineering acquisition strategies

- Application of MBSE
  - Methodology, Style, and Process
  - Implementation
  - Results, Experiences, Lessons Learned
  - Future Research & Conclusions

Application of MBSE emerging as new standard across DoD acquisitions

Air Force acquisition executive unveils next e-Plane, publishes digital engineering guidebook

Published Jan. 19, 2021

Secretary of the Air Force Public Affairs

DEFENSE

Defense contractors explore new ways to develop systems faster and cheaper

DEPARTMENT OF DEFENSE

DIGITAL ENGINEERING STRATEGY

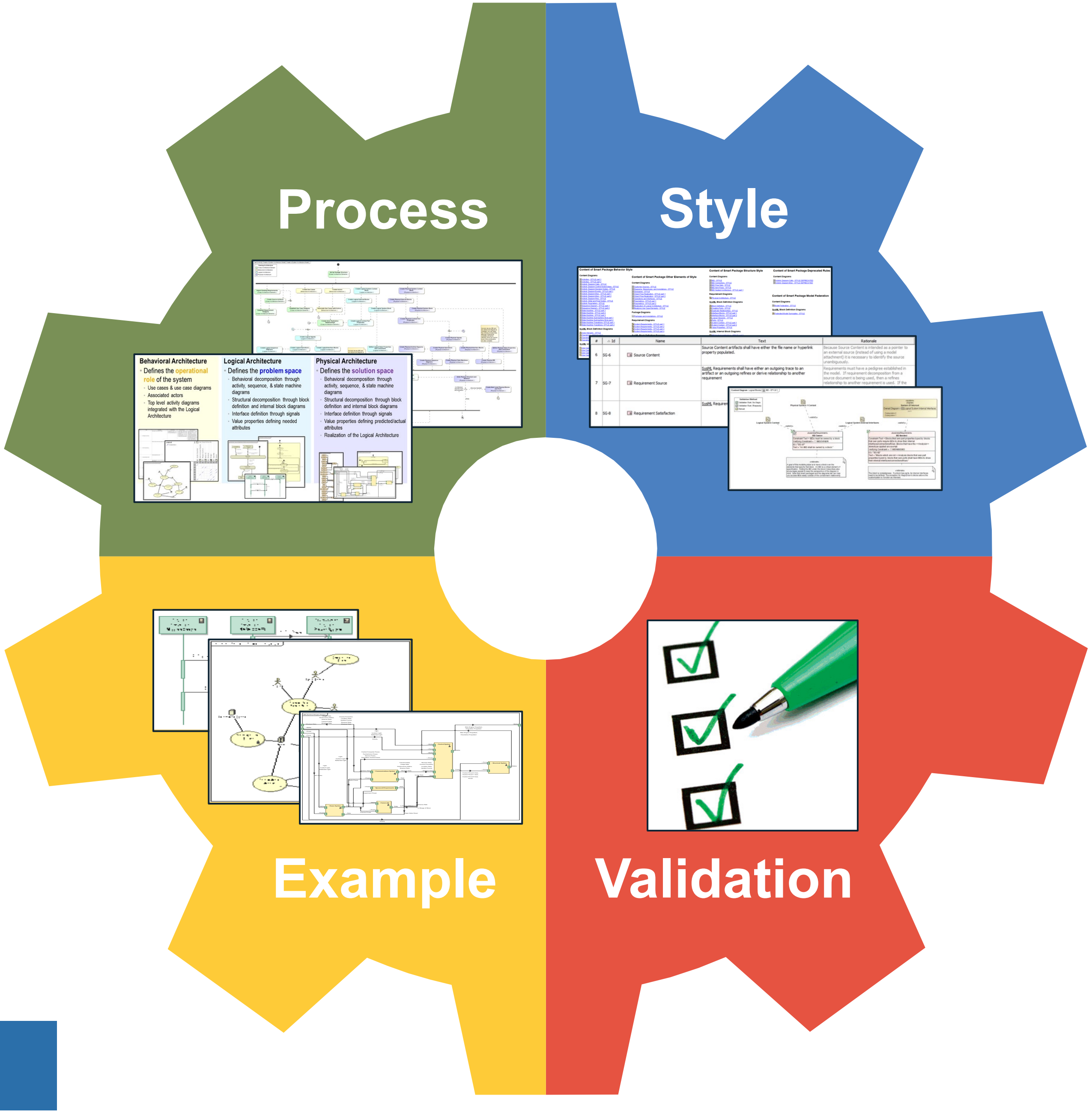
DOD DIGITAL ENGINEERING STRATEGY BASED ON MODEL-BASED SYSTEMS ENGINEERING (MBSE) FOR MILITARY SYSTEM OF SYSTEMS (SOS)

Methodology

Overview

- Based on free & openly published content
  - https://www.saic.com/digital-engineering-validation-tool
  - Swickline, C. and Jugovic, H. (2022), A Data-Centric System Architecture Model Development Process Emphasizing Rapid Tempo and Quality. INCOSE International Symposium, 32: 857-871.
- Processes, methods, and tools:
  - System Architecture Model (SAM) Development Process
  - SAM Style Guide
  - Ranger Rover Example Model
  - SAM DE Validation Suite

No need to reinvent the wheel!



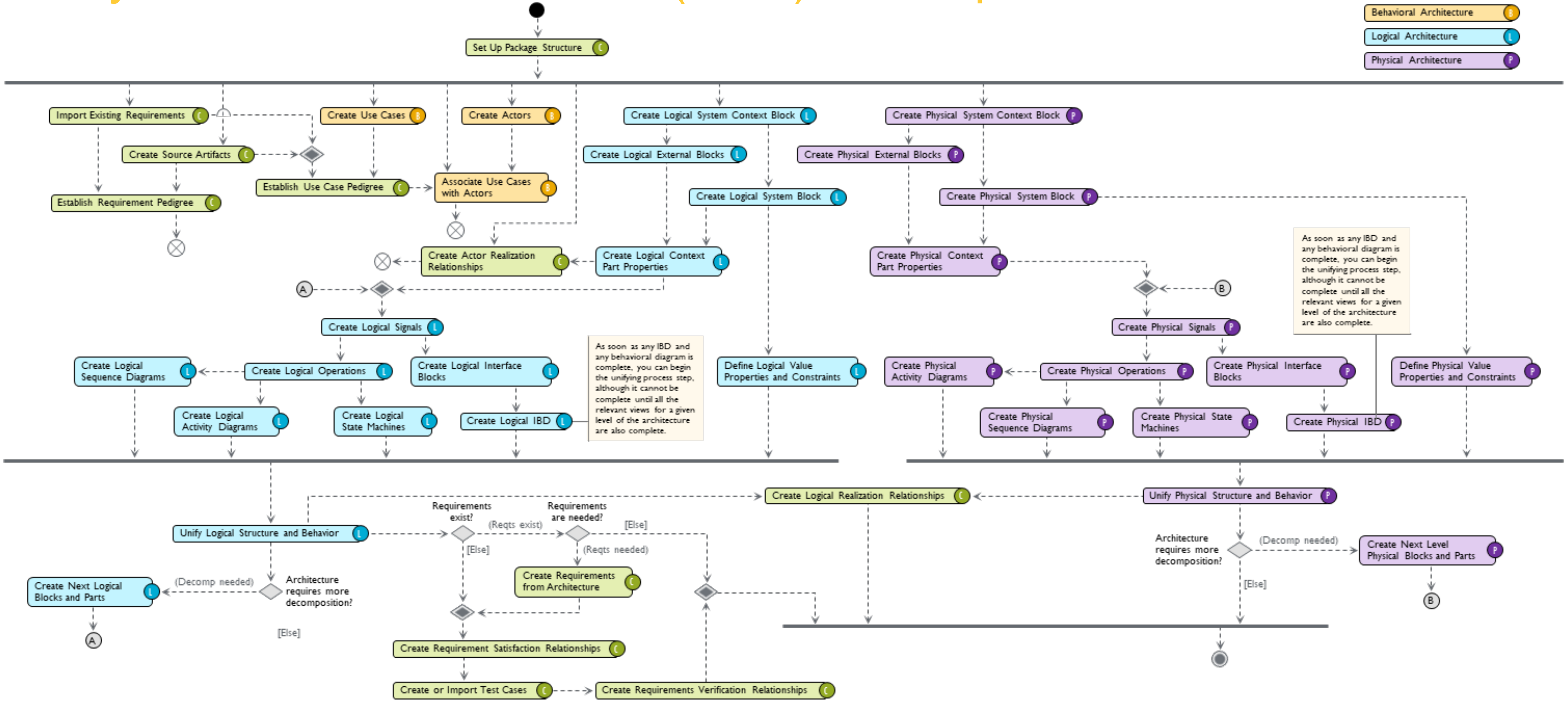
Behavioral Architecture
- Defines the operational role of the system
  - Use cases & use case diagrams
  - Associated actors
  - Top level activity diagrams integrated with the Logical Architecture

Logical Architecture
- Defines the problem space
  - Behavioral decomposition through activity, sequence, & state machine diagrams
  - Structural decomposition through block definition and internal block diagrams
  - Interface definition through signals
  - Value properties defining needed attributes

Physical Architecture
- Defines the solution space
  - Behavioral decomposition through activity, sequence, & state machine diagrams
  - Structural decomposition through block definition and internal block diagrams
  - Interface definition through signals
  - Value properties defining predicted/actual attributes
  - Realization of the Logical Architecture

Methodology

System Architecture Model (SAM) Development Process



Owning Architecture:

- Cross-Architecture Element C
- Behavioral Architecture B
- Logical Architecture L
- Physical Architecture P

Set Up Package Structure C

Import Existing Requirements C

Create Source Artifacts C

Establish Requirement Pedigree C

Create Use Cases B

Establish Use Case Pedigree C

Create Actors B

Associate Use Cases with Actors B

Create Actor Realization Relationships C

Create Logical System Context Block L

Create Logical External Blocks L

Create Logical System Block L

Create Logical Context Part Properties L

Create Logical Signals L

Create Logical Sequence Diagrams L

Create Logical Operations L

Create Logical Interface Blocks L

Create Logical Activity Diagrams L

Create Logical State Machines L

Create Logical IBD L

As soon as any IBD and any behavioral diagram is complete, you can begin the unifying process step, although it cannot be complete until all the relevant views for a given level of the architecture are also complete.

Define Logical Value Properties and Constraints L

Create Physical System Context Block P

Create Physical External Blocks P

Create Physical System Block P

Create Physical Context Part Properties P

Create Physical Signals P

Create Physical Activity Diagrams P

Create Physical Operations P

Create Physical Interface Blocks P

Define Physical Value Properties and Constraints P

Create Physical Sequence Diagrams P

Create Physical State Machines P

Create Physical IBD P

As soon as any IBD and any behavioral diagram is complete, you can begin the unifying process step, although it cannot be complete until all the relevant views for a given level of the architecture are also complete.

Unify Logical Structure and Behavior L

Create Logical Realization Relationships C

Unify Physical Structure and Behavior P

Requirements exist?

(Reqts exist)

[Else]

Requirements are needed?

[Else]

(Reqts needed)

Create Requirements from Architecture C

Create Requirement Satisfaction Relationships C

Create or Import Test Cases C

Create Requirements Verification Relationships C

Create Next Logical Blocks and Parts L

(Decomp needed)

Architecture requires more decomposition?

[Else]

Architecture requires more decomposition?

(Decomp needed)

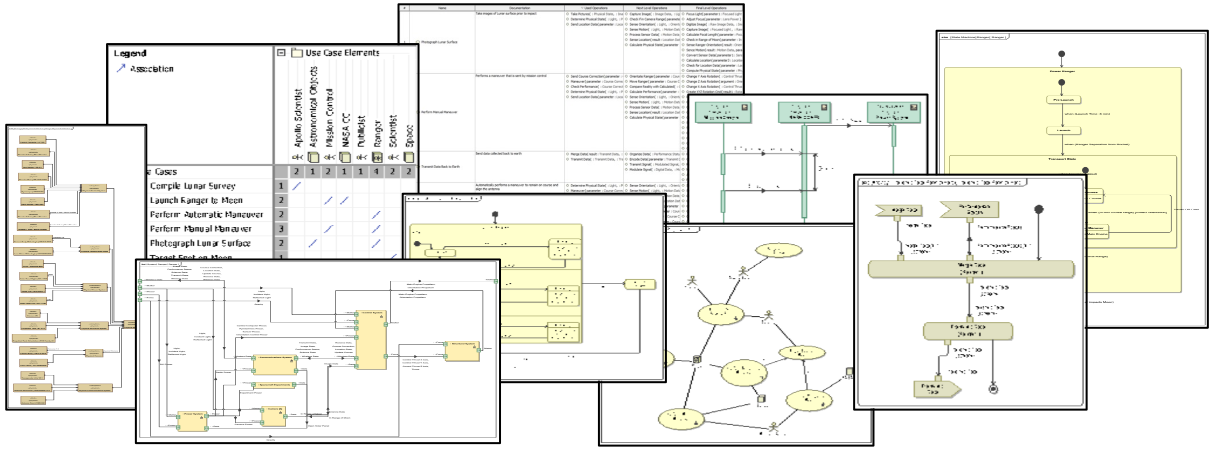
[Else]

Create Next Level Physical Blocks and Parts P

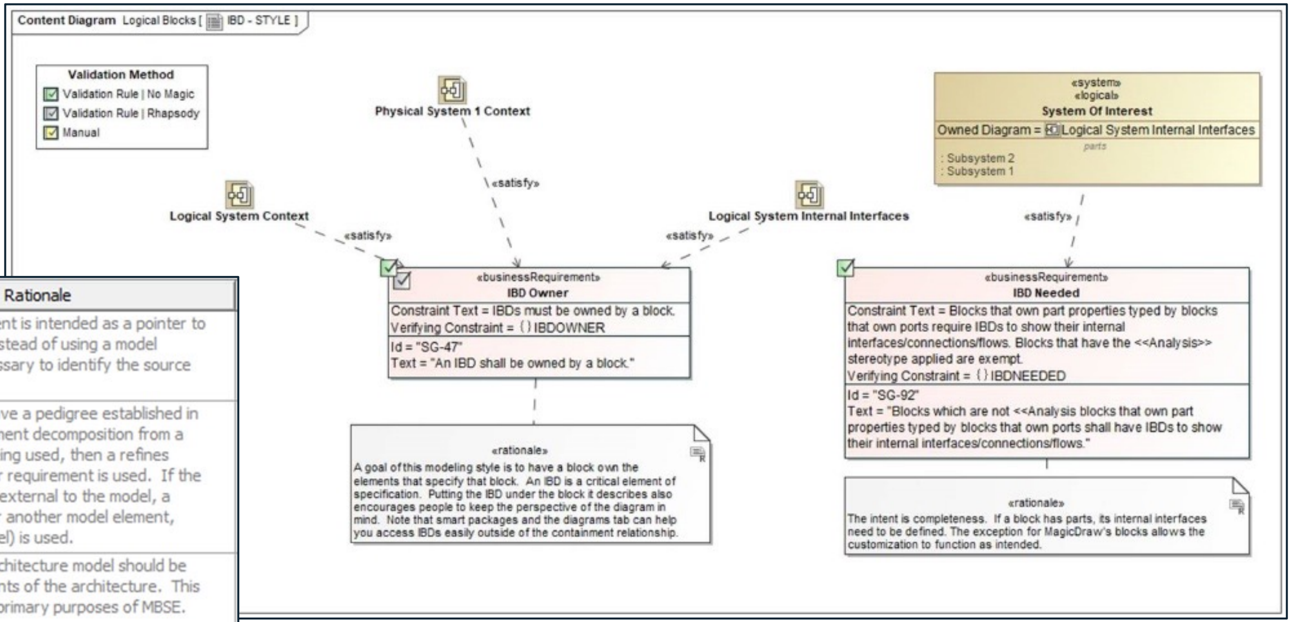
Methodology

Style Guide & Example Model

- Style guide tailors the use of SysML to develop **data-centric** SAMs
- Ranger Rover example model functions as a reference for the style in action



The philosophy is to use the model as a model instead of like a diagraming tool



Constraint Text = IBDs must be owned by a block.
Verifying Constraint = { } IBDOWNER
Id = "SG-47"
Text = "An IBD shall be owned by a block."

A goal of this modeling style is to have a block own the elements that specify that block. An IBD is a critical element of specification. Putting the IBD under the block it describes also encourages people to keep the perspective of the diagram in mind. Note that smart packages and the diagrams tab can help you access IBDs easily outside of the containment relationship.

Constraint Text = Blocks that own part properties typed by blocks that own ports require IBDs to show their internal interfaces/connections/flows. Blocks that have the <<Analysis>> stereotype applied are exempt.
Verifying Constraint = { } IBDNEEDED
Id = "SG-92"
Text = "Blocks which are not <<Analysis blocks that own part properties typed by blocks that own ports shall have IBDs to show their internal interfaces/connections/flows."

The intent is completeness. If a block has parts, its internal interfaces need to be defined. The exception for MagicDraw's blocks allows the customization to function as intended.

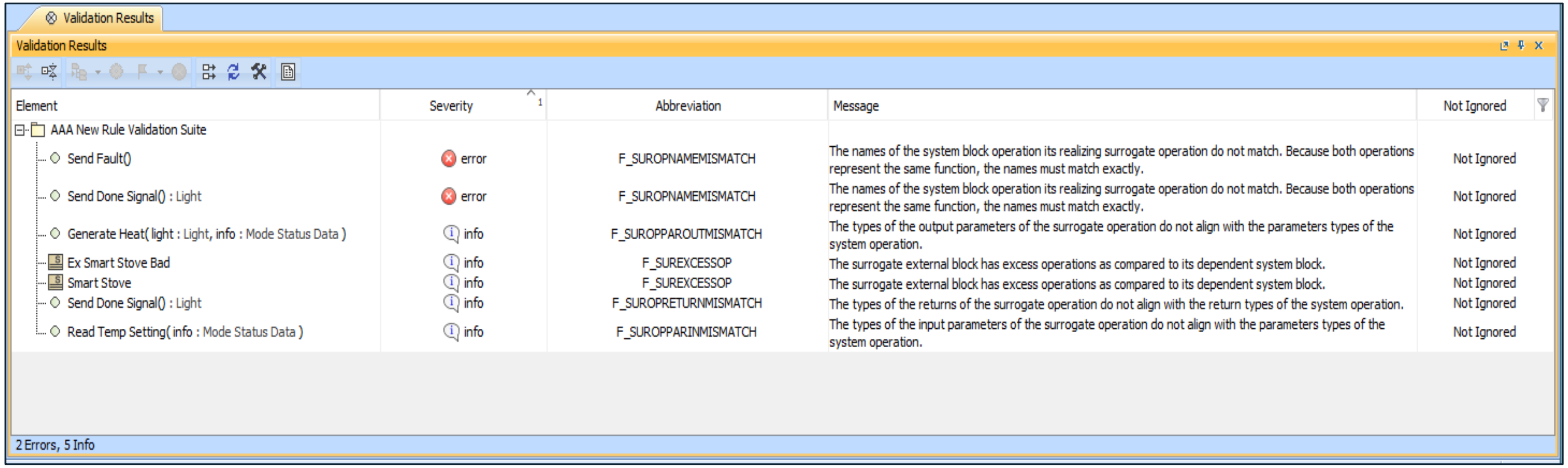
#	△ Id	Name	Text	Rationale
6	SG-6	Source Content	Source Content artifacts shall have either the file name or hyperlink property populated.	Because Source Content is intended as a pointer to an external source (instead of using a model attachment) it is necessary to identify the source unambiguously.
7	SG-7	Requirement Source	SysML Requirements shall have either an outgoing trace to an artifact or an outgoing refines or derive relationship to another requirement	Requirements must have a pedigree established in the model. If requirement decomposition from a source document is being used, then a refines relationship to another requirement is used. If the requirement source is external to the model, a trace to an artifact (or another model element, such as a mission model) is used.
8	SG-8	Requirement Satisfaction	SysML Requirements shall be satisfied by a model element.	Requirements in an architecture model should be satisfied by the elements of the architecture. This analysis is one of the primary purposes of MBSE.

Methodology

Validation Suite

- Automated validation…
  - Improves quality & rigor
  - Increases the tempo for SAM development
  - Facilitates MBSE training
  - Reinforces consistency across SAMs

- SAIC Validation Suite v2.0…
  - Provides 228 unique validation rules
  - Minimizes customizations (only 2 simple unique stereotypes required)
  - Openly published freeware
    - https://www.saic.com/digital-engineering-validation-tool



Element	Severity	Abbreviation	Message	Not Ignored
AAA New Rule Validation Suite				
Send Fault()	error	F_SUROPNAMEMISMATCH	The names of the system block operation its realizing surrogate operation do not match. Because both operations represent the same function, the names must match exactly.	Not Ignored
Send Done Signal() : Light	error	F_SUROPNAMEMISMATCH	The names of the system block operation its realizing surrogate operation do not match. Because both operations represent the same function, the names must match exactly.	Not Ignored
Generate Heat(light : Light, info : Mode Status Data)	info	F_SUROPPAROUTMISMATCH	The types of the output parameters of the surrogate operation do not align with the parameters types of the system operation.	Not Ignored
Ex Smart Stove Bad	info	F_SUREXCESSOP	The surrogate external block has excess operations as compared to its dependent system block.	Not Ignored
Smart Stove	info	F_SUREXCESSOP	The surrogate external block has excess operations as compared to its dependent system block.	Not Ignored
Send Done Signal() : Light	info	F_SUROPRETURNMISMATCH	The types of the returns of the surrogate operation do not align with the return types of the system operation.	Not Ignored
Read Temp Setting(info : Mode Status Data)	info	F_SUROPPARINMISMATCH	The types of the input parameters of the surrogate operation do not align with the parameters types of the system operation.	Not Ignored

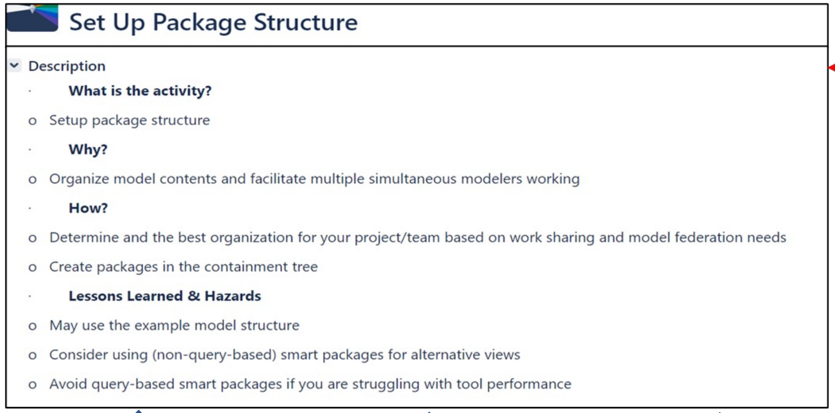
2 Errors, 5 Info

Use the model as a model! Outsource the tedious model review to the computer, freeing up engineers

Application

Agile Backlog Planning

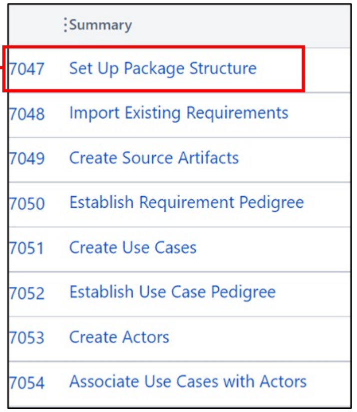
- SAM Development Process Steps used as basis for workflow and backlog development in Jira
- Allowed early planning of architecture development and conformance to the validation rules and validation suite
- Allowed the following to be tracked and reported through Jira:
  - Overall completion status of model segments
  - Open issues
  - Blocked tasks across teams



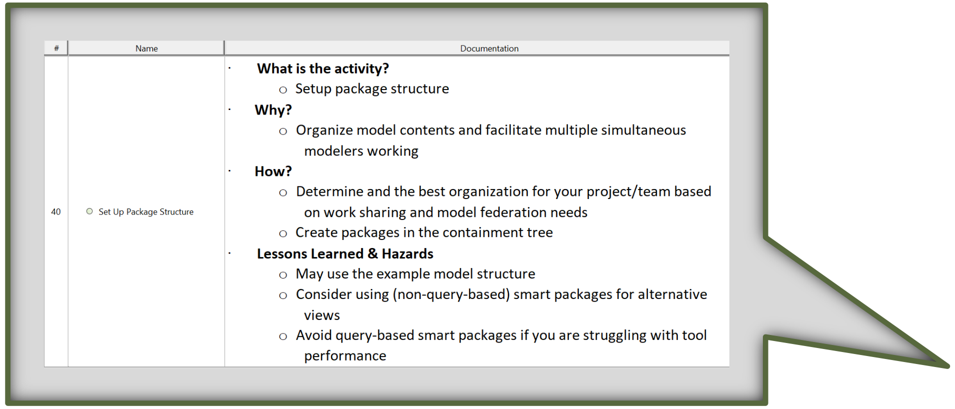
Set Up Package Structure

Description

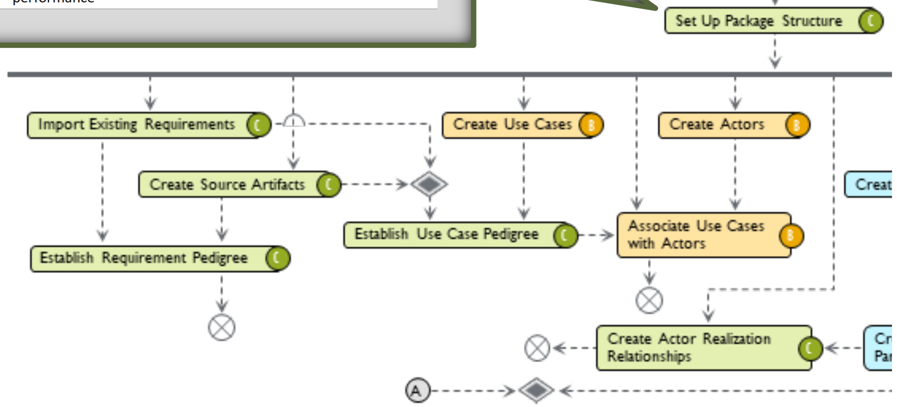
- **What is the activity?**
  - Setup package structure
- **Why?**
  - Organize model contents and facilitate multiple simultaneous modelers working
- **How?**
  - Determine and the best organization for your project/team based on work sharing and model federation needs
  - Create packages in the containment tree
- **Lessons Learned & Hazards**
  - May use the example model structure
  - Consider using (non-query-based) smart packages for alternative views
  - Avoid query-based smart packages if you are struggling with tool performance



	Summary
7047	Set Up Package Structure
7048	Import Existing Requirements
7049	Create Source Artifacts
7050	Establish Requirement Pedigree
7051	Create Use Cases
7052	Establish Use Case Pedigree
7053	Create Actors
7054	Associate Use Cases with Actors



#	Name	Documentation
40	Set Up Package Structure	**What is the activity?** Setup package structure **Why?** Organize model contents and facilitate multiple simultaneous modelers working **How?** Determine and the best organization for your project/team based on work sharing and model federation needs Create packages in the containment tree **Lessons Learned & Hazards** May use the example model structure Consider using (non-query-based) smart packages for alternative views Avoid query-based smart packages if you are struggling with tool performance



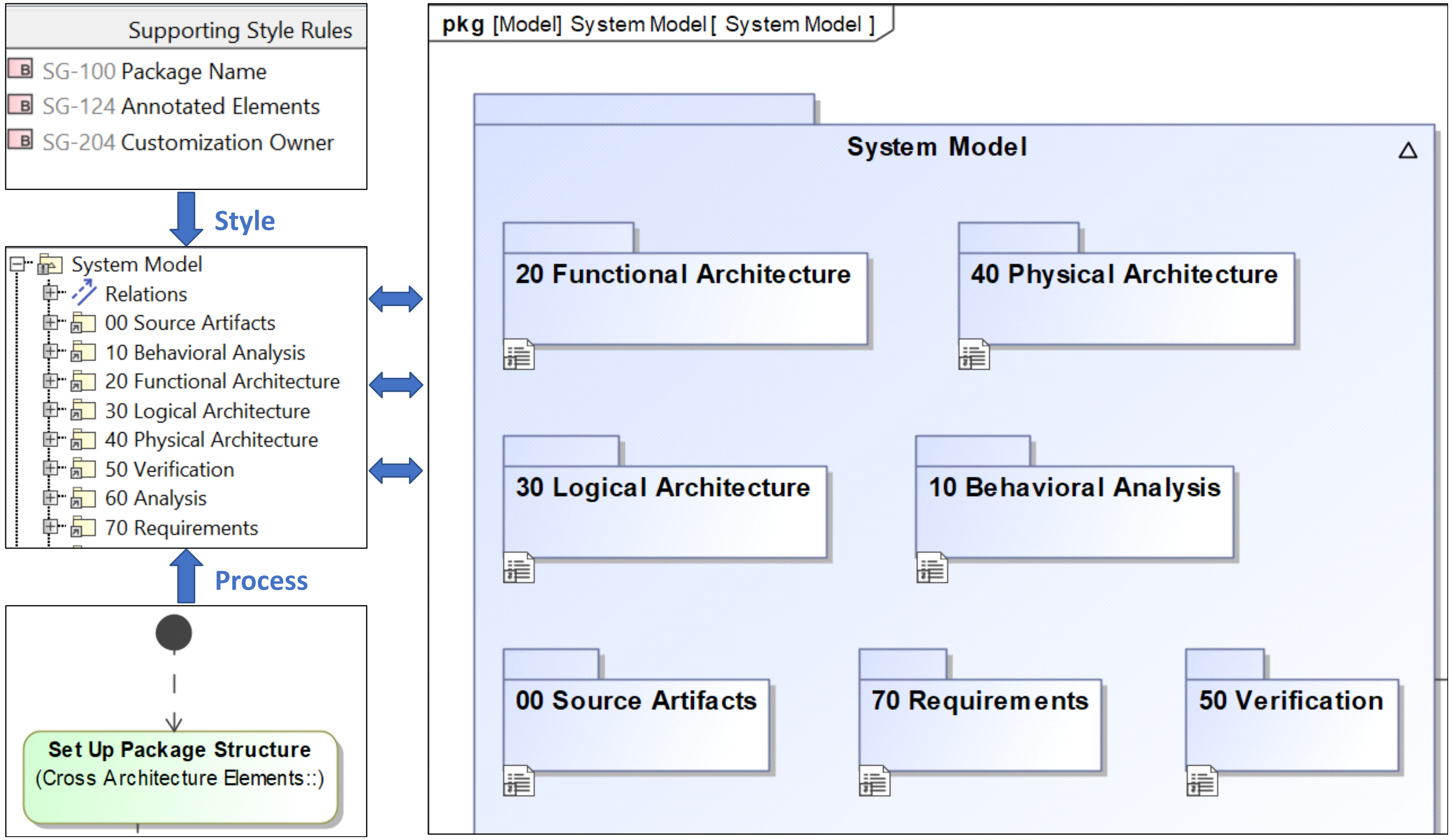
Set Up Package Structure
Import Existing Requirements
Create Use Cases
Create Actors
Create Source Artifacts
Creat
Establish Use Case Pedigree
Associate Use Cases with Actors
Establish Requirement Pedigree
Create Actor Realization Relationships
Cr
Pa
A

Increased task visibility and collaboration across teams and alignment to SAM process

Application

Model Organization & Use Case Development

- Example model structure supports development by multiple engineers
- Figure provides an abridged view of our model which contained:
  - 13 top-level packages
  - Total of 649 packages by CDR
- Maintained logical decomposition of package elements for ease of readability and useability by various stakeholders



Supporting Style Rules

- SG-100 Package Name
- SG-124 Annotated Elements
- SG-204 Customization Owner

Style

- System Model
  - Relations
  - 00 Source Artifacts
  - 10 Behavioral Analysis
  - 20 Functional Architecture
  - 30 Logical Architecture
  - 40 Physical Architecture
  - 50 Verification
  - 60 Analysis
  - 70 Requirements

Process

Set Up Package Structure
(Cross Architecture Elements::)

Model organization and structure enables parallel development

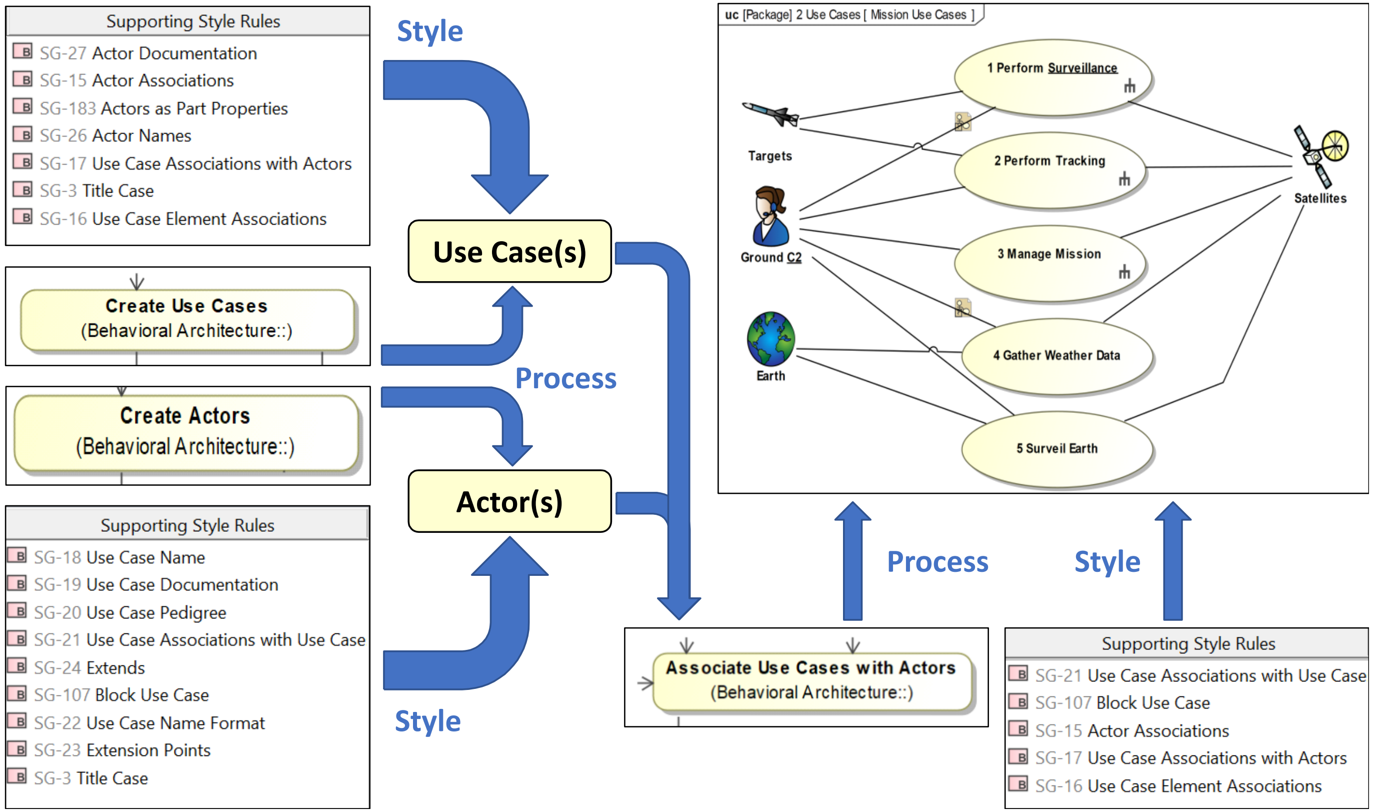
Application

Model Organization & Use Case Development

- All actors and use cases included documentation such that the utility of each element was understood

- Leveraged metachain based table to present use cases along with their documentation, pre-conditions, and basic flow of events per SAM development process

- Structured expressions to query model for information proved more useful than manual diagraming



Supporting Style Rules

- SG-27 Actor Documentation
- SG-15 Actor Associations
- SG-183 Actors as Part Properties
- SG-26 Actor Names
- SG-17 Use Case Associations with Actors
- SG-3 Title Case
- SG-16 Use Case Element Associations

Supporting Style Rules

- SG-18 Use Case Name
- SG-19 Use Case Documentation
- SG-20 Use Case Pedigree
- SG-21 Use Case Associations with Use Case
- SG-24 Extends
- SG-107 Block Use Case
- SG-22 Use Case Name Format
- SG-23 Extension Points
- SG-3 Title Case

Supporting Style Rules

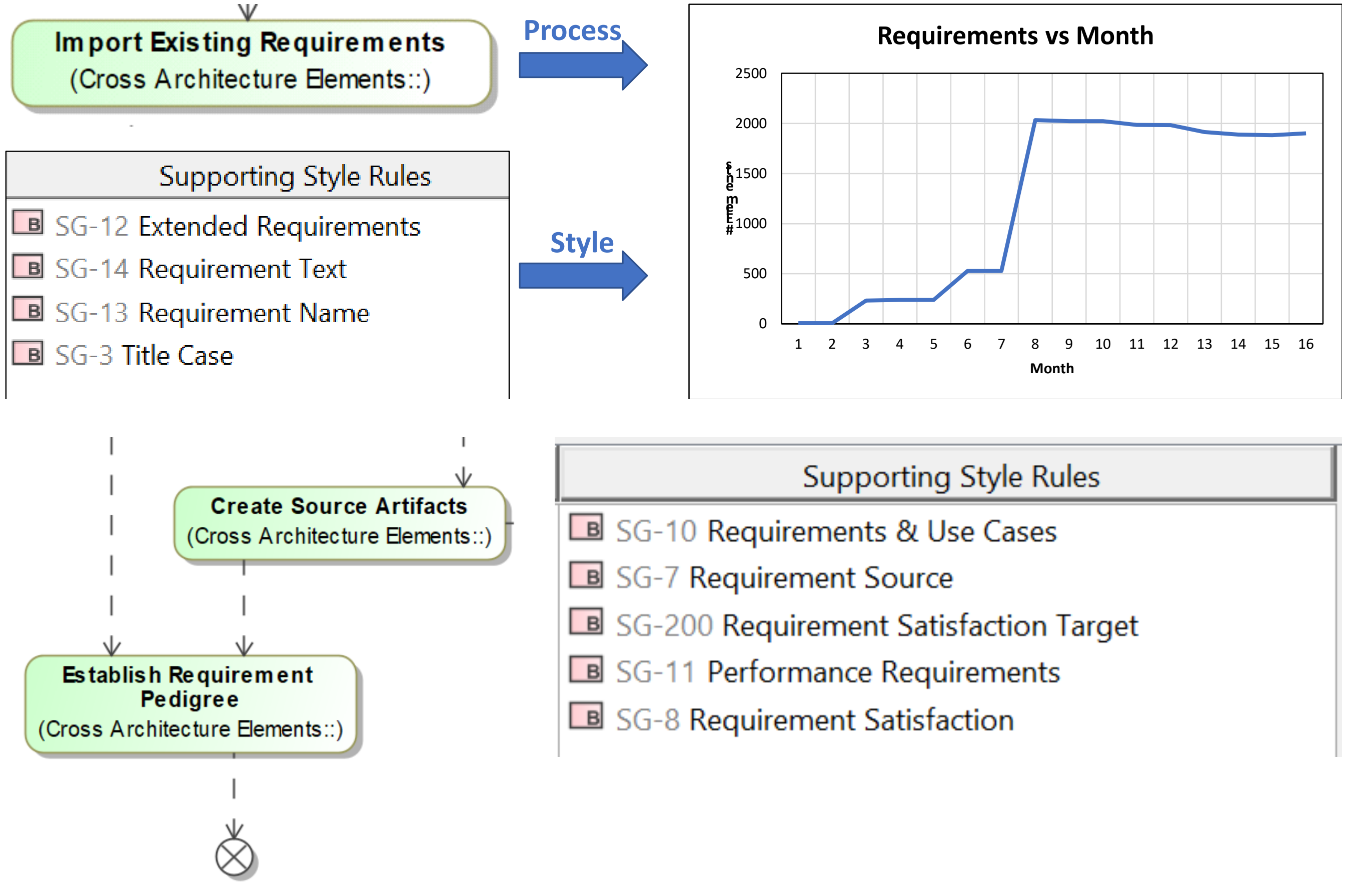
- SG-21 Use Case Associations with Use Case
- SG-107 Block Use Case
- SG-15 Actor Associations
- SG-17 Use Case Associations with Actors
- SG-16 Use Case Element Associations

System operational context defined through use case development to enable functional architecting

Application

Architecture Development

- Imported customer requirements and source artifacts into model
  - Represented sources of information such as documents, open-source content, subject matter experts, and analyses
- Established requirement pedigree to show origin of requirements to unfamiliar stakeholders
  - Generated top to bottom Requirements Verification Traceability Matrices (RVTM)
- ~2000 requirements were imported or created natively in the model over the course of development to CDR


Import Existing Requirements
(Cross Architecture Elements::)
Process
Style
Create Source Artifacts
(Cross Architecture Elements::)
Establish Requirement Pedigree
(Cross Architecture Elements::)
Requirements vs Month
2500
2000
1500
1000
500
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
Month

Supporting Style Rules

- B SG-12 Extended Requirements
- B SG-14 Requirement Text
- B SG-13 Requirement Name
- B SG-3 Title Case

Supporting Style Rules

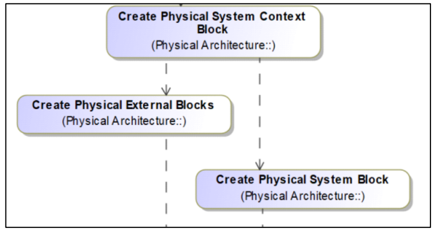
- B SG-10 Requirements & Use Cases
- B SG-7 Requirement Source
- B SG-200 Requirement Satisfaction Target
- B SG-11 Performance Requirements
- B SG-8 Requirement Satisfaction

MBSE methodology for requirements engineering ensured traceability and requirement satisfaction

Application

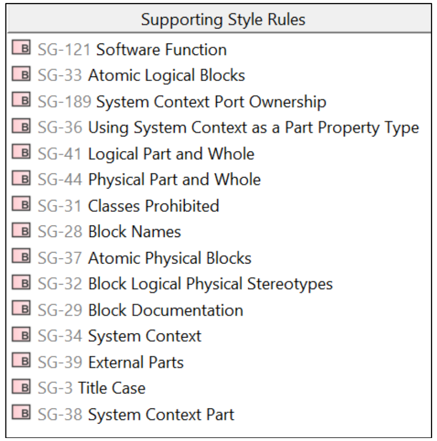
Architecture Development

- Over 800 blocks generated
  - 32 blocks in the logical architecture
  - 116 blocks in the physical architecture
  - Modeled System of Interest (SOI) and external blocks for each interfacing system to our SOI which included:
    - System Context
    - System
    - Subsystem
    - Components
- Block Definition Diagrams (BDDs) used to capture architecture hierarchy and helped in defining the Internal Block Diagrams
  - Internal Block Diagrams (IBDs) used to model the structural configuration of the system and activity diagrams (AD) used to describe system behavior



Create Physical System Context Block
(Physical Architecture::)
Create Physical External Blocks
(Physical Architecture::)
Create Physical System Block
(Physical Architecture::)

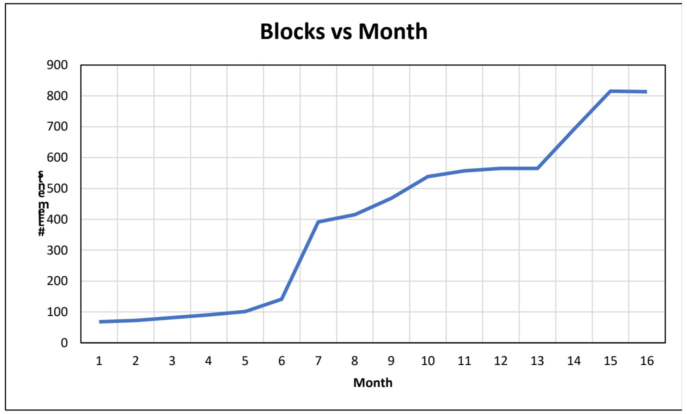
Process



Supporting Style Rules

- SG-121 Software Function
- SG-33 Atomic Logical Blocks
- SG-189 System Context Port Ownership
- SG-36 Using System Context as a Part Property Type
- SG-41 Logical Part and Whole
- SG-44 Physical Part and Whole
- SG-31 Classes Prohibited
- SG-28 Block Names
- SG-37 Atomic Physical Blocks
- SG-32 Block Logical Physical Stereotypes
- SG-29 Block Documentation
- SG-34 System Context
- SG-39 External Parts
- SG-3 Title Case
- SG-38 System Context Part

Style

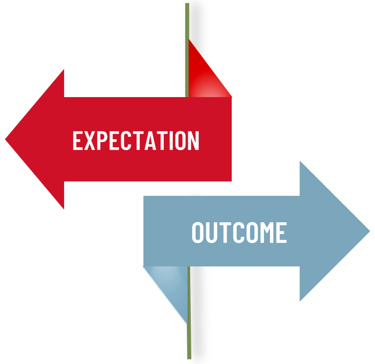


Block Growth Showed Structural and Programmatic Complexity

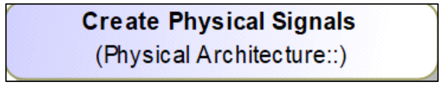
Application

Architecture Development

- Signals were used on IBDs and ADs to consistently describe system interfaces
- Signals were largely used within the physical architecture to represent software messages, hardware electrical interfaces
- Maintained the signals using packages associated with each subsystem

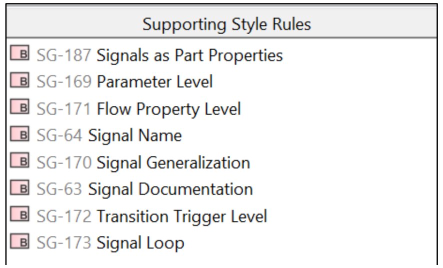


- Validation suite threw errors for missing signals
  - Prompted realignment of structural and behavioral descriptions within the model for that interface
- Sharp increase in signal count when ~430 physical signals were created to represent each unique electrical signal wire
- Learned that a common library including signals would have been easier to use and maintain



Create Physical Signals
(Physical Architecture::)

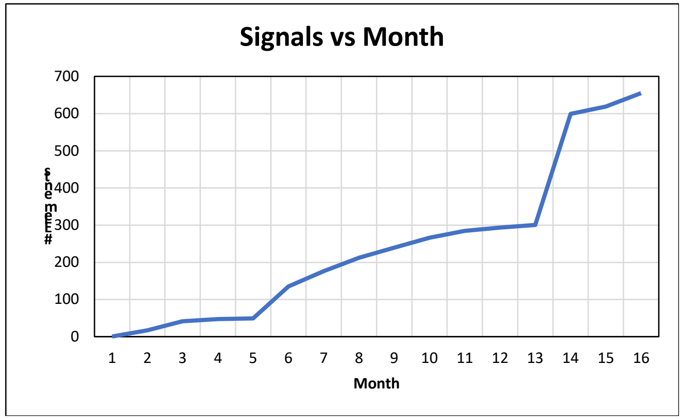
Process



Supporting Style Rules

- SG-187 Signals as Part Properties
- SG-169 Parameter Level
- SG-171 Flow Property Level
- SG-64 Signal Name
- SG-170 Signal Generalization
- SG-63 Signal Documentation
- SG-172 Transition Trigger Level
- SG-173 Signal Loop

Style

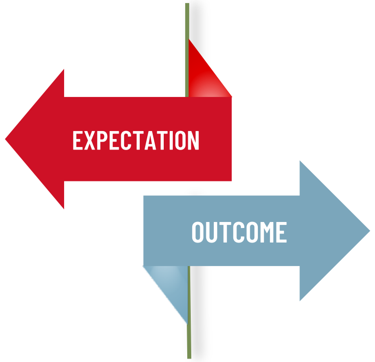


Signal growth showed increased logical, physical and wiring interface definition

Application

Architecture Development

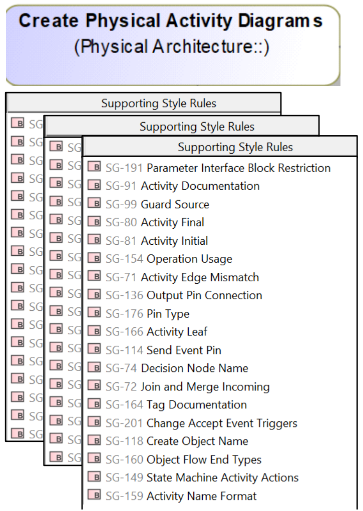
- Initially used Activities/Call Behavior Actions to model system functions instead Operations/Call Operation Actions
- Replaced Call Behavior Actions with Call Operation Actions
  - Describes the use of functions
  - Assigned Activities to the Operations’ methods
  - Decomposed modeled functionality



EXPECTATION

OUTCOME

- Methodology encourages functions to be modeled as Operations owned by the Blocks which perform those functions
  - Validation suite development team later added new validation rule to prevent rework for future teams
- Sharp increase in Call Operation Actions throughout development when new validation rule was introduced



Create Physical Activity Diagrams
(Physical Architecture::)

Supporting Style Rules

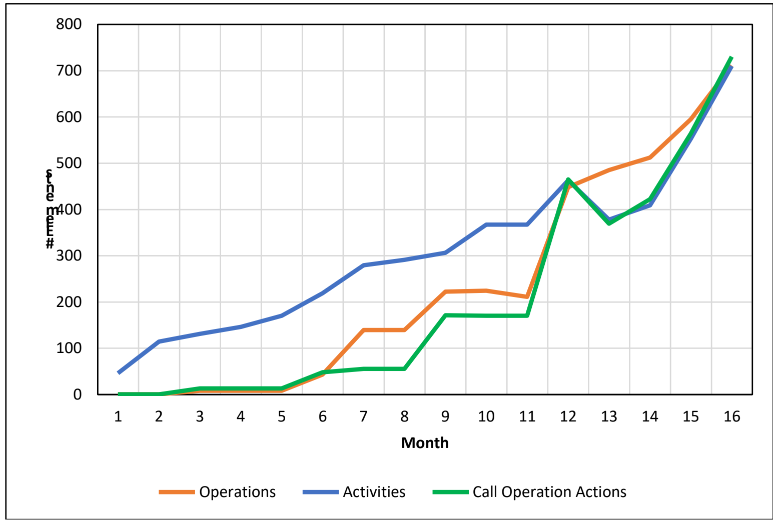
Supporting Style Rules

Supporting Style Rules

- SG-191 Parameter Interface Block Restriction
- SG-91 Activity Documentation
- SG-99 Guard Source
- SG-80 Activity Final
- SG-81 Activity Initial
- SG-154 Operation Usage
- SG-71 Activity Edge Mismatch
- SG-136 Output Pin Connection
- SG-176 Pin Type
- SG-166 Activity Leaf
- SG-114 Send Event Pin
- SG-74 Decision Node Name
- SG-72 Join and Merge Incoming
- SG-164 Tag Documentation
- SG-201 Change Accept Event Triggers
- SG-118 Create Object Name
- SG-160 Object Flow End Types
- SG-149 State Machine Activity Actions
- SG-159 Activity Name Format

Process

Style



Data Reflects the Growth in System Functional Architecture Over Time

Application

Architecture Development

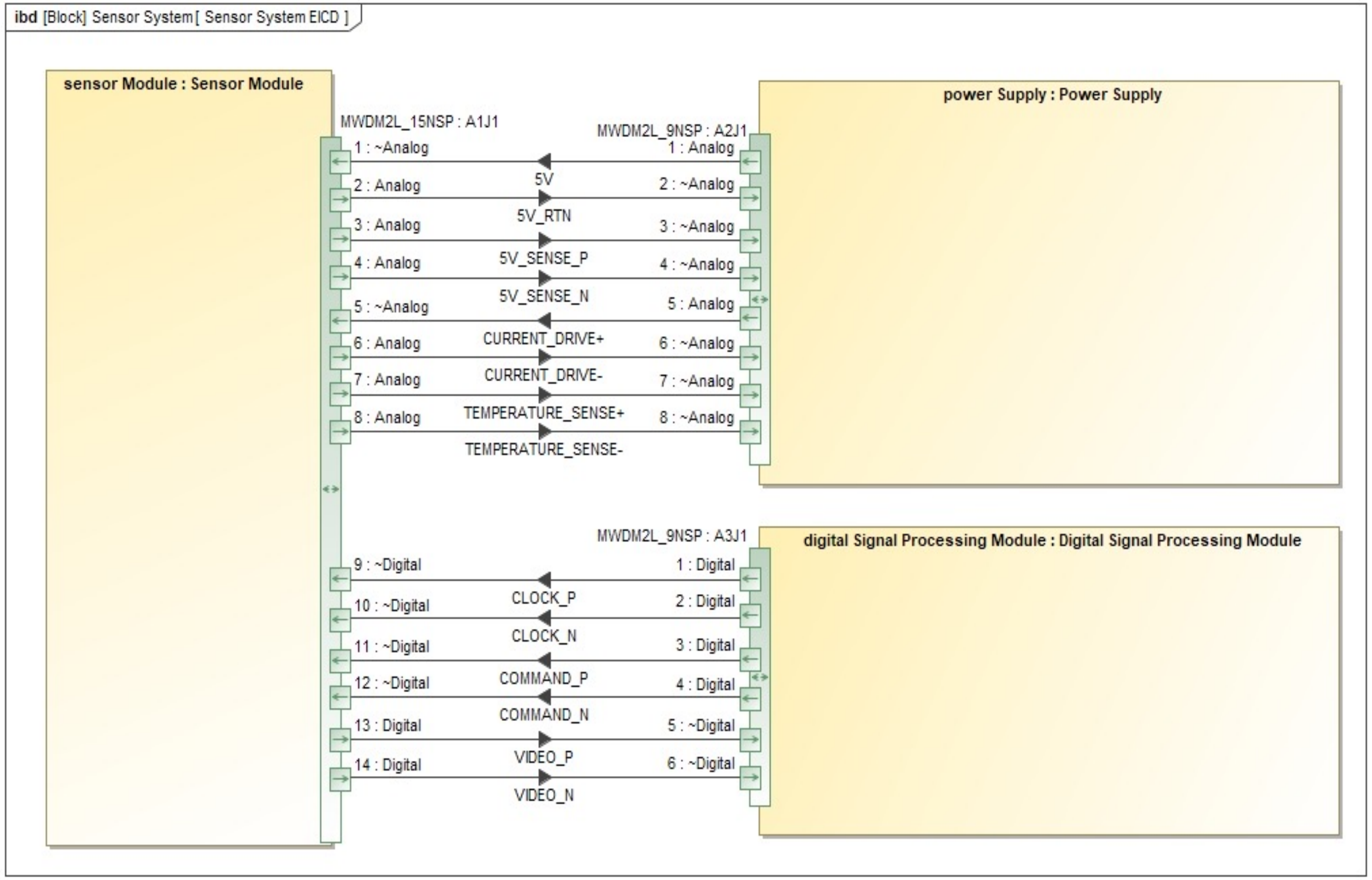
- Modeled the configuration of components and the interfaces between them on physical IBDs

- Applied a nested ports method to show pin level electrical interfaces
  - Developed in accordance with the style guide rules and validation suite

- 62 electrical connectors and 297 electrical signals defined for the sensor by CDR

- Electrical Interface Control Documents (EICDs) developed to coordinate with a critical vendor responsible for providing 23 electrical connectors and 133 electrical signals

Pin level interfaces developed using IBDs

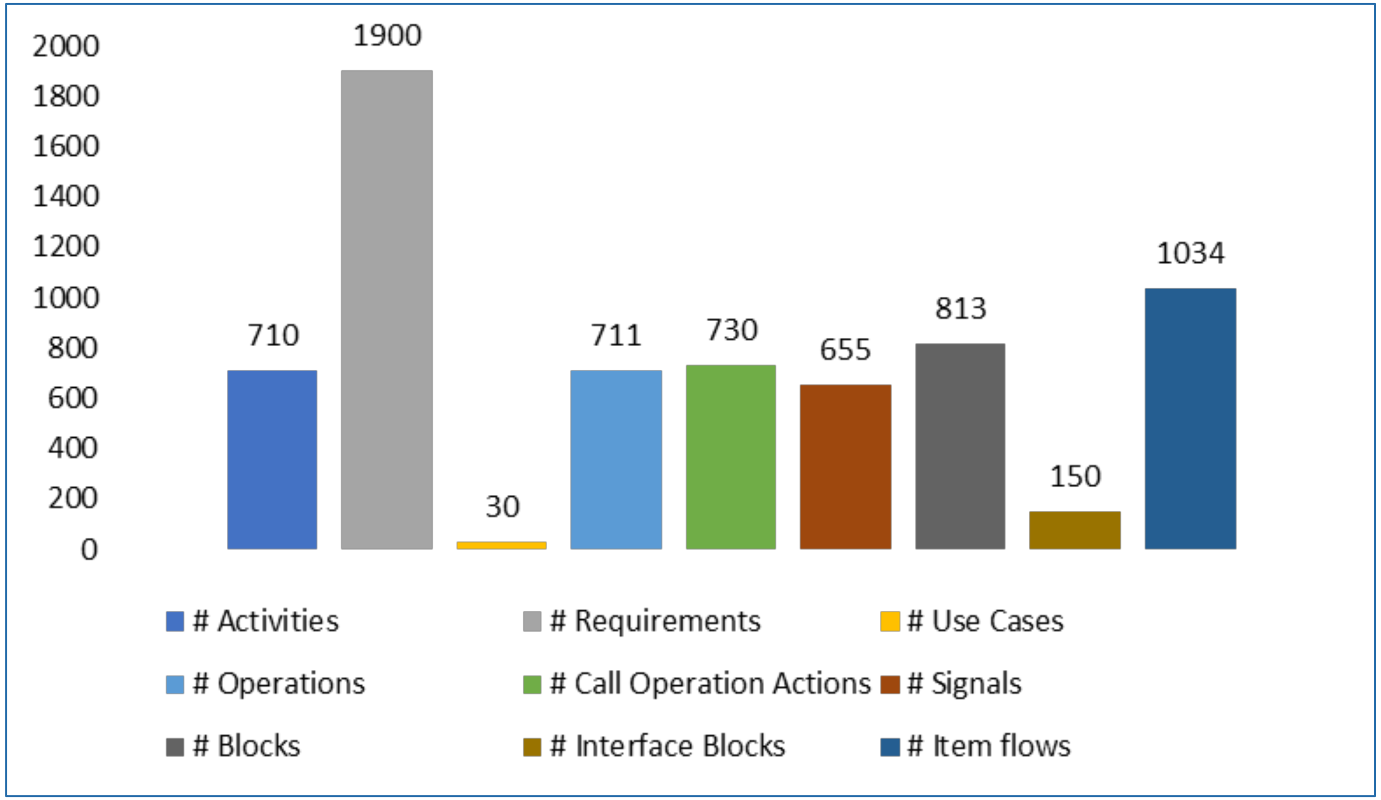


Results

- ~80,000 unique elements generated within primary system model
  - Common elements shown in figure

- ~260,000 elements generated in total across the project usages

- >38 contractually deliverable items completed based on model content
  - Verification Cross Reference Matrices (VCRMs)
  - Interface Control Documents (ICDs)
  - Concept of Operations (CONOPS)
  - Functional architecture
  - Product breakdown structure
  - Derived or used data directly from the model

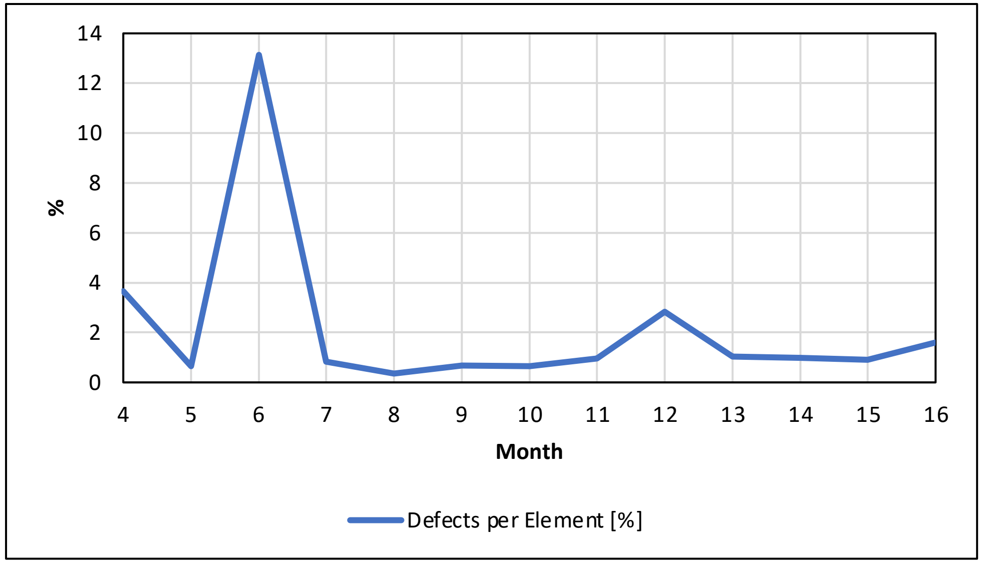


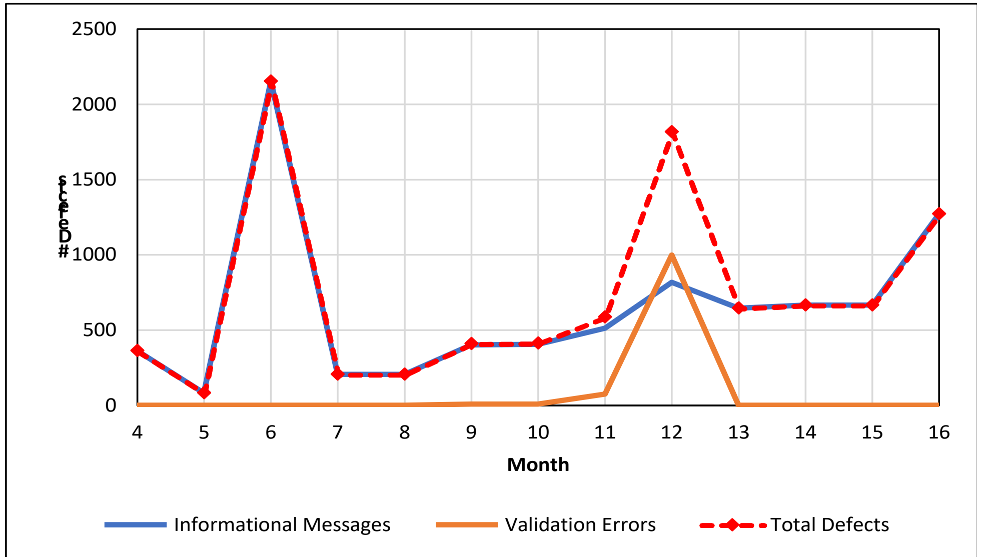
Nearly **80,000** elements generated following SAM process and validated!

Results

- Development process, style guide, and associated validation suite allowed our team to develop a system architecture model satisfying program needs
- Low percentage of model defects relative to model elements shows a well-formed model
- CDR model delivery passed validation with zero errors and 1272 informational warnings
- Delivered technical content as part of an integrated model baseline that historically has been delivered as disparate static documentation

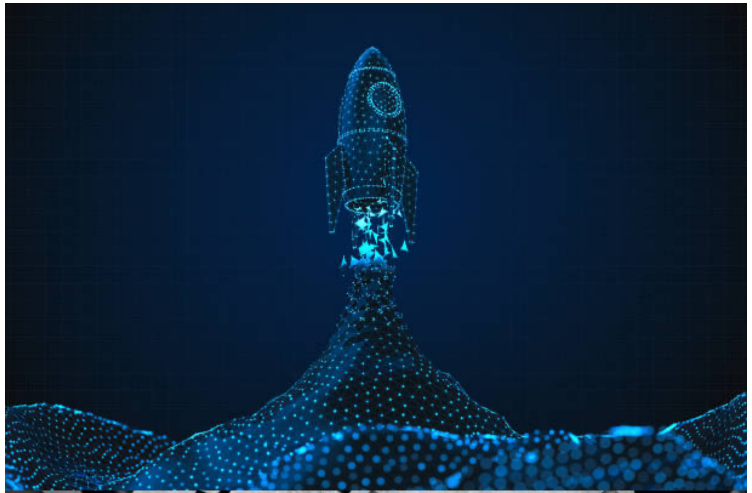
Total defects in validation caused by informational warnings. **Zero errors** remained at CDR





Experience Applying the Methodology

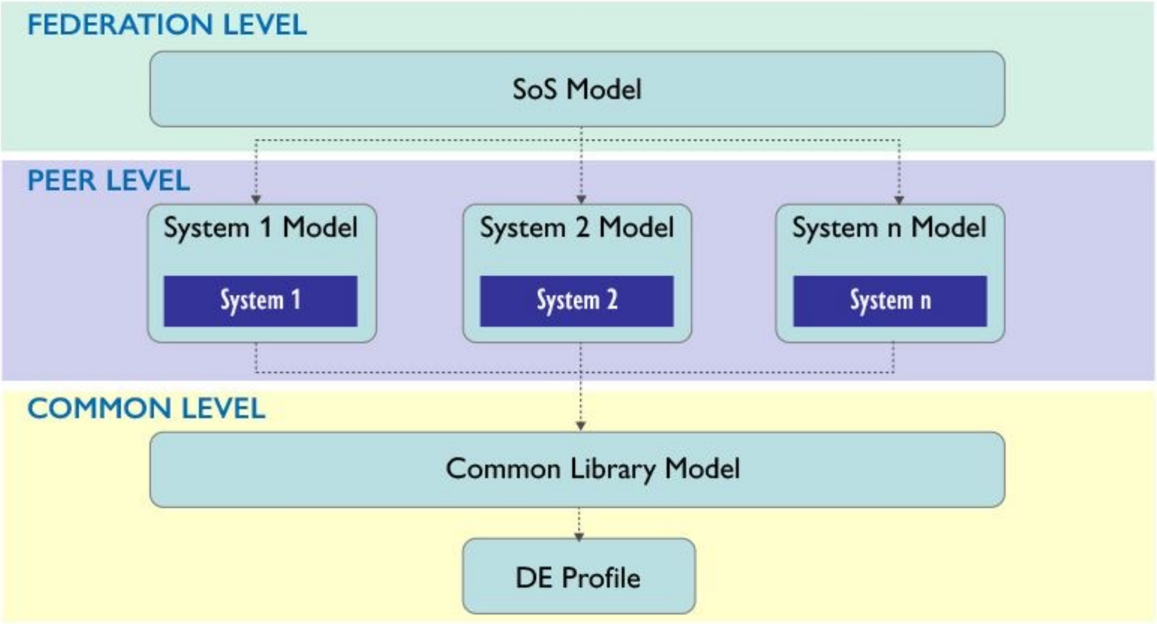
- This methodology offers some key value for users
  - Model completeness, consistency, and correctness
  - Training of best practices through development (leveraging validation suite)
- This methodology is currently being adopted across multiple programs with some waivers.
- Consistent use of this methodology will streamline model development.
  - This methodology require upfront effort and planning



With careful pre-planning and roadmap, this methodology provides means to develop complete and consistent **system model**

Future Research & Conclusions

- Future Research
  - Model Federation
  - Common Libraries
  - Direct Connection of Data Through Analytical Models (Digital Thread)



Developing System of System Models using SAIC tools
Create Digital Thread

Chris Madariaga
christopher.madariaga@rtx.com

Ahmad Bashir
ahmad.bashir@rtx.com

Chris Swickline
chris.r.swickline@saic.com

SAIC Digital Engineering
https://www.saic.com/digital-engineering-validation-tool

33rd Annual INCOSE
international symposium

hybrid event

Honolulu, HI, USA
July 15 - 20, 2023

www.incose.org/symp2023

TBD

TBD

- TBD

TBD