A Roadmap for Model-Based Systems Engineering

David Long
Vitech Corporation

A Roadmap for Today

• A Brief Introduction to Systems Engineering
• MBSE Explained – Models, Representations, and an Approach
• Applying MBSE: From Requirements through V&V
  – Defining the boundary
  – Capturing the requirements
  – Defining the system logic
  – Implementing the system logic
  – Testing 1,2,3
• SE Representations: SysML and Beyond (time permitting)
• Closing Thoughts

Please raise questions and offer perspectives as they occur!
A Brief Introduction to Systems Engineering

What is a System?

An integrated composite of people, products, and processes that provide a capability to satisfy a stated need or objective.

A system is a construct or collection of different elements that together produce results not obtainable by the elements alone.

Some examples include

- A set of things working together as parts of a mechanism or an interconnecting network
- A set of organs in the body with a common structure or function
- A group of related hardware units or software programs or both, especially when dedicated to a single application
- A group of celestial objects connected by their mutual attractive forces, especially moving in orbits about a center
System Essentials

• A System ‘is a thing that contains interconnected smaller things, interacts with other things in a larger thing, and does something.’
  – has structure, in a larger structure (‘physical’ context)
  – performs purposeful actions
    • time sequence of collective actions = behavior
  – behavior observed at interfaces (‘functional’ context)

• “… a system is, and a system does …”

What is Systems Engineering?

Systems Engineering is an engineering discipline whose responsibility is creating and executing an interdisciplinary process to ensure that the customer and stakeholder’s needs are satisfied in a high quality, trustworthy, cost efficient and schedule compliant manner throughout a system’s entire life cycle.

INCOSE
Setting the Context:
The Four Primary SE Activities

Three Systems of Interest
Failure to Think Systemically

**NUTRIA...NEWEST OPPORTUNITY**

New fur-bearing animal now being bred in U.S. Offers huge profits to breeders. This fur is next to Mink NOW in price and will soon surpass it. 15 to 20 young per year. Most easily and economically raised animal known. 1/4¢ per day to feed, ANY CLIMATE, disease resistant. The Cabana Marrone strain is obtainable ONLY through Cabana Nutria, Inc. and its authorized dealers and distributors. For free booklet and address of Cabana ranch nearest you write:

CABANA NUTRIA, INC. Dept. 27
636 West Lemon Avenue, Arcadia, California

Failure to Think Systemically

**Blackwater Wildlife Preserve**
Old Model: Waterfall

- Requirements
- Capabilities
- Architecture
- Testing

Stovepiping
It is Very Like a Snake

Systems Engineer’s Dilemma: Complexity and Synchronization

Any change will affect something else
Model-Based Systems Engineering

Systems Engineering: A Practice in Transition

Traditional
- Specifications
- Interface requirements
- System design
- Analysis & Trade-off
- Test plans

Future

Moving from document-centric to model-centric

Reprinted from INCOSE Model-Based Systems Engineering Workshop, February 2010
Model-Based Systems Engineering

Model-based systems engineering (MBSE) is the formalized application of modeling to support system requirements, design, analysis, verification and validation activities beginning in the conceptual design phase and continuing throughout development and later life cycle phases.

Two Ways of Looking at Systems

ANALYTIC

SYNTHETIC

One Integrated Model

Source Requirements Domain

Behavior Domain

Behavior is allocated to physical components

Architecture Domain

Behavior is allocated to physical components

Originating requirements trace to behavior

Verified by

Data

Verified by

Originating requirements trace to physical components

Verified by

Data
Model-Based System Engineering

- Systems engineering paradigm shift
- System model is essential and required
- System model encompasses the system design and specification
- System specifications are complete and consistent
- Model is provided to subsequent engineering teams
Essential Components of MBSE

- A declared metamodel / language
  - Structure and semantics
  - Textual and graphical
  - Explicit, context-free language for communication
  - Problem, solution, and management dimensions

- A process or methodology

- Defined mappings / projections
  - “Fit for purpose” views
  - Documentation and design artifacts
  - Other work products

A Simplified Language for SE
A Simplified Language for SE

REQ → BASIS OF → BEH → ALLOCATED TO → ARCH

BEH → BASED ON → ALLOCATED TO → PERFORMS

What are Views?
Views DO NOT Equal A Model

A Consistent View of Views
seq Thread 1: Product in Inventory

Customers → Geospatial Library

- t1. Make Information Request
- t1. Information Request
- t1. Accept & Format Request
- t1. Get Product From Inventory
- t1. Provide Product To Customer
- t1. Accept Products
- t1. Collection Products

A Consistent View of Views
A Model-Based Approach Supporting “Fit for Purpose” Views

The Automatic Generation of Representations and Work Products

Views are projections of the model. Choose the views that serve the purpose and drive them from a single, integrated model.
Model-Based Systems Engineering

- Formalizes SE practice through the use of models
- Broad in scope
  - Integrates with multiple modeling domains across life cycle from SoS to component
- Results in quality/productivity improvements & lower risk
  - Rigor and precision
  - Communications among system/project stakeholders
  - Management of complexity

An Illustrated Walkthrough from Requirements to V&V

An Old Story Reborn
Model-Based Systems Engineering
Activities Timeline – Top Down

0. Define Need & System Concept
1. Capture & Analyze Orig. Requirements
2. Define System Boundary
3. Capture Originating Architecture Constraints
4. Derive System Threads
5. Derive Integrated System Behavior
6. Derive Component Hierarchy
7. Allocate Behavior to Components
8. Define Internal Interfaces
9. Select Design
10. Perform Effectiveness & Feasibility Analyses
11. Define Resources, Error Detection, & Recovery Behavior
12. Develop Validation Requirements/Validation Plans
13. Generate Documentation and Specifications

Activity bars represent movement of “center of gravity” of systems engineering team. Concurrent engineering is assumed.

MBSE Activities Timeline – Reverse Engineering

1. Define System Boundary
2. Capture Interfaces
3. Capture Component Hierarchy
4. Derive As-Built Behavior of Components
5. Aggregate to As-Built System Behavior
6. Derive As-Built System Threads
7. Derive As-Built System Requirements

Find the top, then modify top-down.

8. Update System Boundary
9. Select Design
10. Perform Effectiveness & Feasibility Analyses
12. Develop Test Plans
13. Generate Documentation and Specifications
A Roadmap for Our Example

• Defining the boundary
• Capturing the requirements
• Defining the system logic
• Implementing the system logic
• Testing 1,2,3

WWI Image Management System
Integrated Systems Engineering Process

Source Documents

LEVEL 1

REQ BEH ARCH Docs

LEVEL 2

REQ BEH ARCH Docs

LEVEL n

REQ BEH ARCH Docs

Level Of Detail

Define the System Boundary
Boundary 1:
A Collector-Owned System

Boundary 2:
A Customer-Owned System
Requirements (Sources)

- System Concept Paper
- Executive Order
- Concept of Operations
- Statement of Work
- Vendor Package/Contract
- Preliminary Specification
- Change Request Trade Study Report
- Standards (MIL-STD or Commercial)
- Meeting Minutes’
- Business Plan
- Market Analysis

Desired Characteristics of Requirement Statements

- Necessary – remove it if the statement is not needed
- Implementation independent – state what is required, not how the requirement is met
- Unambiguous – generates a common understanding
- Complete – can be understood in isolation
- Singular – addresses one thought
- Feasible – is inherently possible
- Verifiable – can confirm the requirement is satisfied
- Correct – properly expresses the stakeholder expectation
- Conforming – conforms in look and feel to organizational standards

Additional information available in INCOSE Guide for Writing Requirements
Desired Characteristics of Requirement Sets

• Complete – represents the full definition of the stakeholder expectations
• Consistent – reconciled and individual statements do not conflict with one another
• Feasible – can be satisfied by a solution that is obtainable within life cycle constraints
• Bounded – establish the system scope and do not address subjects outside that scope
• Structured – organized such that sub-sets of requirement statements can be identified

Additional information available in INCOSE Guide for Writing Requirements

Requirements Analysis

• Decompose originating requirements to single testable statements
• Capture test requirements as validation requirements
• Extract/decompose requirements – do not edit
• Be careful to not change the meaning of the requirements
• Provide traceability from the source document to the parent originating requirements
• Provide traceability from each parent requirement to its children.
• Prefer hierarchy traceability but flat file organization of requirements is acceptable
Requirements for the Geospatial Library

1.0 SCOPE

This source specification establishes the basis for the performance, design, development, and test requirements for an Geospatial Library.

This Geospatial Library is intended to serve as a means to demonstrate the use of automated system engineering support tools. As defined, this documentation system accepts requests for imagery information, determines the best way for the system to respond to the request, and then provides the requested information to the requester. In the process of acquiring the requested information, the system may generate tasking orders for a set of imagery data collectors.

The mission of the Geospatial Library is to provide management of a system of Imagery collectors, from the acceptance of customer requests, through scheduling the collectors, to delivery of the imagery products to the customers.

2.0 APPLICABLE DOCUMENTS

The applicable documents for the Geospatial Library program are:

EIA 182 (System Engineering Standards)

3.0 GENERAL REQUIREMENTS

The Geospatial Library shall provide continuous real-time support to the customers and the collection systems. The system shall be unavailable no more than a total of ten minutes per month.

3.1 Specific Requirements:

3.1.1. The system shall accept information requests from certified customers.

3.1.2. The system shall retain an inventory of previously collected images/products and provide them to users, if appropriate.

3.1.3. The system shall control multiple image collectors and multiple types of image collectors.

3.1.4. The system shall be staffed at a maximum of 25 personnel on any shift.

3.1.5. The system shall provide feedback on the customer's request within twenty four hours.

3.1.6. The system shall provide a means of prioritizing the customer's requests.

3.1.7. The system shall monitor and assess its own performance.

3.2 Specific Requirements:

Accept Requests

The system shall accept information requests from certified customers.

Accept Media Requests

The system shall accept requests from certified customers via any of the following media: 1) Hardcopy Forms; 2) Verbal; 3) Phone-verbal; 4) Phone-electronic file; and 5) PC diskette-electronic file.

Retain Inventory

The system shall provide feedback on the customer's request within twenty four hours.

Control Multiple Collectors

The system shall control multiple image collectors and multiple types of image collectors.

Maximum Staff

The system shall be staffed at a maximum of 25 personnel on any shift.

Provide Feedback

The system shall provide feedback on the customer's request within twenty four hours.

Prioritize Requests

The system shall provide a means of prioritizing the customer's requests.

Monitor and Assess

The system shall monitor and assess its own performance.

Monitor Self Performance

The system shall monitor its own performance.

Assess Self Performance

The system shall assess its own performance.
Requirements: Traceability

Sources

Tool

Model

Concerns

• During the requirements capture and analysis process, it is likely that problems will be found:
  – Unclear or incomplete requirements
  – Contradictory requirements
  – Requirement for unlikely system performance
  – Over- or under-specified requirements
Risks

• Problem which requires a mitigation plan
• Uncertainty of achieving a product or program milestone
• Reasons
  – Budget or schedule
  – Complex technology
  – New designs or concepts
  – Criticality
• Resolution captured by mitigation plan
• Tracked until reduced or resolved
Behavior

- Shows what a system does or appears to do without regard to how (implementation) it does it
- Is represented graphically by a model which integrates the control (functions and constructs) model and the interface (inputs and outputs) model
- Specifies the system logic

A Complete Set of Executable Constructs (Structured Representations)
A Complete Set of Executable Constructs (SysML Representations)

- **SEQUENCE**
- **SELECT**
- **MULTIPLE-EXIT**
- **ITERATE**
- **LOOP**
- **REPLICATE**

Functions & Items

- **Function**
  - A process that transforms inputs into outputs
  - An action taken by the system or one of its elements
  - Represented by a verb or verb-noun pair

- **Function Enablement**
  - Enabled upon completion of the function prior to it in the sequencing

- **Function Execution**
  - Requires enablement and triggering, if a trigger is defined

- **Triggers**
  - Item that provides a control role
  - Defined by a **triggers** relationship
  - Shown with a double arrowhead (EFFBD) or without annotation (activity diagram)

- **Data Stores**
  - Item that does not provide a control role
  - Defined by an **input to** relationship
  - Shown with a single arrowhead (EFFBD) or with “optional” annotation (activity diagram)
Representing System Behavior

**EFFBD**

**Activity Diagram**

**Sequence**

**N2 Diagram**

Identifying Use Cases
Building and Using Threads

- Define distinct classes of threads based on system I/O and/or conditions.
- Start with one simple thread per class of system input and/or conditions.
- Preserve each Thread (for thread testing, concept of operations, etc.)

1a. Derive threads

1b. Partition threads

2. Integrate threads to define integrated system behavior

Thread One: Image in Inventory

CUSTOMER

Request Image
Accept Request

SYSTEM

Check Product Inventory
Get Product From Inventory
Provide Product To Customer

CUSTOMER
Thread Two: Image Not in Inventory

CUSTOMER
- Request Image
- Accept Request

SYSTEM
- Check Product Inventory
- Get Product From Inventory
- Put Product In Inventory
- Provide Product To Customer
- Accept & Format Product
- Task Collectors

CUSTOMER


Thread Integration

CUSTOMER
- In Inventory

SYSTEM
- COLLECTOR
Thread Integration

Integrated System Behavior
(Sample EFFBD)
Allocate Behavior to Components

- Allocate behavior to components
- This activity defines the interfaces between system components
- May require more than one iteration to get feasible and desirable interfaces
- May require iteration between deriving the component hierarchy and allocating the behavior to that hierarchy
Relating the Functional and Physical Models

Partition the System Behavior among Components for a “Trial Allocation”

- Determine partitioning strategy(s). Examples:
  - Function
  - Existing components
  - Performance
  - Response time
  - Testing implications
  - Interface complexity
  - Technology risks
  - Future performance requirements
  - Future technologies
- Associate behaviors to components
  - Explore many alternative partitions
  - Identify resulting internal interfaces
  - (i.e., Items flow between functions associated with different Components)
How Do We Know When We Are Done?

• Decompose behavior in sufficient detail:
  – If every function can be uniquely allocated to only one component
    **STOP DECOMPOSING!**
  – If a function cannot be uniquely allocated to only one component
    **DECOMPOSE IT FURTHER!**

Full Traceability
(Bi-directional)

**REQUIREMENT:**
The system shall provide previously collected images/products, if appropriate.

**FUNCTION:**
The system shall provide previously collected images/products, if appropriate.

**IS THE BASIS OF**

**ARCHITECTURE**

**IS ALLOCATED TO**

**IS BASED ON**

**PERFORMS**
Verification and Validation

• Verification
  – shows that all the requirements produced through the system design process are indeed satisfied within the physical instantiation of the system and its components

• Validation
  – demonstrates that the system is the “right” solution—one that is usable, useful and fulfills the customer’s needs

“We fail more often because we solve the wrong problem than because we get the wrong solution to the right problem.” - Russell Ackoff
Composing Verification Requirements

- What is the Verification Requirement’s objective?
- What method will be employed to verify the Verification Requirement’s target system/component requirements?
- What are the environmental conditions associated with the Verification Requirement?
- Are there any special conditions associated with the Verification Requirement?
- What are the success criteria for the Verification Requirement?
The Testing Dilemma

As Designed

As Reported

As Understood

As Tested

Test Plans

• Must be developed
  – For selected design configurations
  – To identify test support components
    • Integration aids
    • Test aids
    • Facilities
  – To identify cost/schedule drivers
# System Testing Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Functional Testing</td>
<td>Test conditions are set up to ensure that the correct outputs are produced, based upon the inputs of the test conditions. Focus is on whether the outputs are correct given the inputs (also called black box testing).</td>
</tr>
<tr>
<td>Structural Testing</td>
<td>Examines the structure of the system and its proper functioning. Includes such elements as performance, recovery, stress, security, safety, and availability. Some of the less obvious elements are described below.</td>
</tr>
<tr>
<td>Performance</td>
<td>Examination of the system performance under a range of nominal conditions; ensures system is operational as well.</td>
</tr>
<tr>
<td>Recovery</td>
<td>Various failure modes are created and the system’s ability to return to an operational mode is determined.</td>
</tr>
<tr>
<td>Interface</td>
<td>Examination of all interface conditions associated with the system’s reception of inputs and sending of outputs.</td>
</tr>
<tr>
<td>Stress Testing</td>
<td>Above-normal loads are placed on the system to ensure that the system can handle them; these above-normal loads are increased to determine the system’s breaking point; these tests proceed for a long period of time in an environment as close to real as possible.</td>
</tr>
</tbody>
</table>

## Test Paths:
- 1, 2, 4, 6
- 1, 2, 5, 6
- 1, 3, 6

---

# Test Threads

![Test Threads Diagram]

1. Accept and Format Request
2. Check Product Inventory
3. Accept and Format Collector Products
4. Prioritize Request
5. Collect Data
6. Collect the Mix
7. Accept and Format Collector Products
8. Put Product in Inventory

---

The Testing Problem

- The systems that we develop are usually dynamic; thus, the system specification should be dynamically consistent/executable at the system level
  - Can it be achieved by confirming that the individual subsystems are executable?
    - No – modern control theory shows that executable subsystems do not provide an executable system
  - Can we use simulation of the design?
    - Yes – but common dynamic verification simulators are not exact representations of the system being specified
  - If not executable, what happens?
    - Achievement of dynamic consistency is left to integration and test teams, or
    - The system fails to meet dynamic needs

How Do We Do It Today?

- Give the simulation team a copy of preliminary system specification
- They interpret it and build the simulator
- If the delivered simulator results seem reasonable, it is accepted
- Problem?
  - We dynamically verified a system that is different than specified. It is actually the test team’s interpretation of the system specification.
  - Integration and test team must finish the systems engineering by plugging gaps
How Do We Solve the Problem?

• Use model-based systems engineering
• Automatically generate the simulator and the specification from the same model
• Do not allow the simulator to be “modified” without changing the specification

DISCUSSION / Q&A REGARDING OUR WALKTHROUGH
SE Representations:
SysML and Beyond

View and Viewpoints

Multiple views;
How many viewpoints?

http://en.wikipedia.org/wiki/Multiview_orthographic_projection
Criteria for Diagram Choice

• Who is your audience?
• What do they want/need to see?
• What do you want/need to tell them?

Notable References for Additional Information

• [www.omgsysml.org](http://www.omgsysml.org) - official SysML specification published by OMG
  – v 1.3 released June 2012

Hierarchy Diagrams

- Classic representation of relationships between several layers of objects
  - Most frequently used to represent composition or traceability
- No special symbology
- Suitable for any audience
Requirements Diagrams

• Expand classic hierarchy diagram to represent key aspects of requirements
  – Graphical format often complemented with tabular representation
• Usage largely limited to providing context for a few requirements

Key Semantics of Requirements Diagrams

• <<containment>> - partitioning of a complex requirement without adding meaning
• <<deriveReqt>> - more detailed requirement arrived at via analysis
• <<satisfy>> - assertion that a model element satisfies the requirement
• <<verify>> - reference to a test case or verification aspect

Additional information available in chapter 13 of A Practical Guide to SysML
Additional Requirement Views

**Tables**

- **Level of Detail:** High
- **Audience:** General
- **Content:** Requirement properties and relationships
- **Use:** Requirement lists; traceability matrices; verification matrices

**Specifications**

- **Level of Detail:** High
- **Audience:** General (including contract officers)
- **Content:** System or subsystem requirements
- **Use:** Textual representation of requirements – generally compliant with a specific document format – used for milestone reviews and transmission across contractual boundaries

VIEWS DEPICT BEHAVIOR

- Make Information Request
- Accept & Format Request
- Get Product from Inventory
- Provide Product to Customer
- Accept Product
An Integrated Picture of Behavioral Views

Concepts reflected
- Composition
- Control / Structure
- Triggering
- Data Flow
- Allocation

What about
- audience?
- level of detail?

Use Case Diagrams

- Describe the functionality of a system from the user perspective
  - Elicit requirements or bridge to system threads
- Represents actors, blocks, and use cases
- Complemented with
  - Preconditions
  - Postconditions
  - Primary flow
  - Exception flows
- Suitable for most audiences
Key Semantics of Use Case Diagrams

- Actors
- Blocks
- Use cases
  - Inclusion
  - Extension
  - Classification

Sequence Diagrams

- Emphasize interaction between collaborating parts of a system
  - Particularly powerful when used in understanding logical threads
  - Includes lightweight representation of behavioral constructs
- Suited to a general audience
- Well understood (and often requested) by software engineers
Key Semantics of Sequence Diagrams

- Blocks and lifelines
- Events
- Messages
- Interaction operators (constructs)
  - Par – parallel
  - Alt – exclusive or
  - Loop

Additional information available in chapter 10 of *A Practical Guide to SysML*

Enhanced Function Flow Block Diagrams (EFFBDs)

- Represent the full specification of system behavior
  - Control flow
  - Data flow
  - Resources
  - Allocation
- Structured equivalent to SysML activity diagram
- Best suited when dealing with broader audiences
Key Semantics of EFFBDs

- **Functions**
  - Sequence
  - Select
  - Parallel
  - Loop
  - Iterate
  - Replicate

- **Data flow**
  - Triggers
  - Data

- **Resources**

Activity Diagrams

- Reflect the most complete SysML representation of behavior
  - Control flow
  - Data flow
  - Allocation

- Best suited when dealing with software audiences or working at the design level
Key Semantics of Activity Diagrams

- Activities
- Control
  - Join
  - Fork
  - Decision
  - Loops
- Data flow
  - Required
  - Optional
- Swim lanes

Additional information available in chapter 9 of *A Practical Guide to SysML*

N2 (“N-squared”) Diagrams

- Represent the logical data flow for a system or segment
  - Highlight what’s there and what’s not
  - Identify implied interfaces
- Physical variants represent components and interfaces / links
- Suitable for general audiences
  - More often used for analysis (including clustering) than communication
Key Semantics of N2 Diagrams

- Functions shown on the diagonal
- Inputs shown in the column
- Outputs shown in the row

IDEF0 Diagrams

- Present an integrated picture of the inputs, control, outputs, and mechanisms (ICOM)
- Includes visual diagnostics of inconsistencies across behavioral decomposition
- Falling out of favor in the general community
  - Still frequently used by process engineers
Key Semantics of IDEF0 Diagrams

- Functions
- ICOM
  - Input – left
  - Control – top
  - Output – right
  - Mechanism – bottom
- Tunneling

Additional information available in National Institute of Standards and Technology (NIST), FIPS PUB 183

Additional Behavioral Views

DoDAF OV-1

Level of Detail: Low
Audience: General
Content: General context often with lightweight composition, triggering, and allocation
Use: High-level contextual introduction to describe operational boundaries and align system vision

State Transition Diagram

Level of Detail: Medium
Audience: System and software engineers
Content: System states and the corresponding transitions
Use: Insight into the system by taking an orthogonal look at behavior

Simulation Timeline

Level of Detail: Medium
Audience: General
Content: True performance aspects of specified behavior
Use: Debugging system logic; analysis of performance characteristics
An Integrated Picture of Physical Views

Hierarchical View

Block Definition

Physical View

Hierarchy

Block Definition

Physical N2

Concepts

Composition

Connections

Inheritance

PHYSICAL CHARACTERISTICS SPECTRUM

More composition
Less connectivity

Less composition
More connectivity
Block Definition Diagrams

- Represent the composition structure of blocks
  - Extend classic physical hierarchies with greater richness
  - Often include block roles and characteristics
- Best suited for system/software engineers and subject matter experts

Key Semantics of Block Definition Diagrams

- Blocks
  - Operations
  - Values
  - Ports
- Roles
- Multiplicity
- Composition vs reference (“in, but not of”)

Additional information available in chapter 7 of *A Practical Guide to SysML*
**Internal Block Diagrams**

- Specify logical or physical connectivity of blocks
  - Extend classic block diagrams with ports, directionality, and data
- Best suited for system/software engineers and subject matter experts

**Key Semantics of Internal Block Diagrams**

- Blocks with roles
- Connections
- Ports
  - in
  - out
  - inout
  - Required and provided interfaces

Additional information available in chapter 7 of *A Practical Guide to SysML*
Additional Physical Views

**DoDAF SV-1**

**Level of Detail:** Medium  
**Audience:** General  
**Content:** General context with lightweight composition and connectivity (logical and physical)  
**Use:** High-level contextual introduction to describe system boundaries and align system vision

**Physical Hierarchy**

**Level of Detail:** Low  
**Audience:** General  
**Content:** Multi-level specification of system composition  
**Use:** In-depth hierarchical presentation of parts list

Physical Views, cont.

**Block Definition Diagram**  
**Level of Detail:** High  
**Audience:** System and software engineers  
**Content:** System inheritance model  
**Use:** Detailed representation of any system inheritance and corresponding characteristics; software class diagram

**Block Definition Diagram (Structure)**  
**Level of Detail:** High  
**Audience:** System/software engineers and subject matter experts (SMEs)  
**Content:** Physical composition often including block roles and characteristics  
**Use:** Detailed, multi-level design representation of system composition and corresponding physical characteristics

**Block Definition Diagram (Classification)**  
**Level of Detail:** High  
**Audience:** System and software engineers  
**Content:** System inheritance model  
**Use:** Detailed representation of any system inheritance and corresponding characteristics; software class diagram
Additional Physical Views, cont.

**Physical N2 Diagram**

**Level of Detail:** Low  
**Audience:** General  
**Content:** Single-level composition with corresponding logical (interface) or physical (link) connections  
**Use:** High-level identification of connections; clustering analysis

**Interface Block and Physical Block Diagrams**

**Level of Detail:** Medium  
**Audience:** Not software engineers or SysML zealots  
**Content:** Composition with logical or physical connectivity  
**Use:** Specification of logical or physical connections; boundary definition; insight into external connections

VIEWS DEPICT MUCH MORE
A Few Additional Representations

Level of Detail: High
**Audience:** Systems engineers and SMEs
**Content:** System parameter definition
**Use:** Mathematical specification of key system parameters

Level of Detail: Low
**Audience:** General
**Content:** Object names and interrelationships
**Use:** Contextual view of objects of interest with no implied meaning

Level of Detail: Low
**Audience:** General
**Content:** Fusion content
**Use:** Single page focused illustration; often measurement / management

One Conceptual Progression of SE Representations

0. Define Need & System Concept
1. Capture & Analyze Orig. Requirements
2. Define System Boundary
3. Capture Originating Architecture Constraints
4. Derive System Threads
5. Derive Integrated System Behavior
6. Derive Component Hierarchy
7. Allocate Behavior to Components
8. Define Internal Interfaces
9. Select Design
10. Perform Effectiveness & Feasibility Analyses
11. Define Resources, Error Detection, & Recovery Behavior
12. Develop Validation Requirements/Validation Plans
13. Generate Documentation and Specifications
Representation Overload – Remember the Concept of “Fit for Purpose”

A Consistent View of Views
The Trap to Avoid

Disjoint fit-for-purpose views can wreck our project on the rocky shores

Diagrams must be representational tools from the model, not a substitute for a model

Closing Thoughts

The Effective SE Process
Complex Problems, Diverse Groups, and Clear Needs

Key Messages about Systems Engineering

- Understanding what to do in systems engineering is easy
- Doing systems engineering well is difficult
- Managing complexity is a major element of the problem
- Good systems engineering needs:
  - Good systems engineering process,
  - Good tools that support the process,
  - Documented procedures and standards,
  - Good technical management,
  - Good engineers

Automated tools do not do systems engineering... only people do systems engineering
Adding Detail and Clarity through Iterative Design Activities

- Iterate the layers of systems engineering
- Structure behavior to assure operational capacity
- Add resources and/or related behavior
- Add error detection and recovery behavior
- Evaluate the design and perform required trade studies
- Finish validation requirements and test plans
How Do You Know When You Are Done?

<table>
<thead>
<tr>
<th>Process Elements</th>
<th>Completion Criteria</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Originating Requirements</td>
<td>1. Agreement on Acceptance Criteria</td>
</tr>
<tr>
<td>2. Behavior Definition</td>
<td>2. Each function can be allocated to at most one component</td>
</tr>
<tr>
<td>3. Architecture Definition</td>
<td>3. Segment/component specs are complete per requirements documents</td>
</tr>
<tr>
<td>4. Qualification</td>
<td>4. V&amp;V requirements have been traced to test system components</td>
</tr>
</tbody>
</table>

An Integrated, Model-Based Approach to Systems Engineering

- Provides discipline and structure
- Enhances communication
- Increases quality
- Reduces risk
- Ensures convergence through layered approach
- Speeds delivery and enhances agility, especially in the face of change
- Accelerates (radically) the exploration of revisions, alternatives, and variants

Decide how you want to apply the gains from the adoption of MBSE – rapid development, greater quality, or analysis of alternatives
The Age of Systems Challenges.
The Age of Model-Based Systems Engineering?

For Additional Information

David Long
Vitech Corporation
2270 Kraft Drive, Suite 1600
Blacksburg, VA, 24060, USA
1.540.951.3322
dlong@vitechcorp.com
www.vitechcorp.com
www.mbseprimer.com