Abstract

This basic tutorial identifies the elements and benefits of a complete, proven model-based system engineering process, and demonstrates its tailorability and value for project success using vignettes from an information management system and a sample System of Systems (SoS) application. The tutorial illustrates how the model-based system engineering process supports both document-driven and model-based paradigms, whether in top-down, middle-out, or reverse engineering environments. It discusses how to know when each element of the process has been completed, and how to develop and validate functional and physical architectures using executable architectures. The requirement for concurrent engineering, the onion model, and synchronization of models and data are presented.

The participants will be introduced to a flexible system engineering process suitable for system development tasks across the complexity spectrum. In addition to the process description, the tutorial will include a sample solution to illustrate the recommended techniques, views, completion conditions, and products of an MBSE system development methodology. It will also include examples of the development of graphical views commonly used by practitioners of DoDAR and SysML approaches.

This tutorial is focused on highlighting how the use of model-based systems engineering can meet the government requirements for delivering architecture framework products while allowing the engineering organizations (industry and government) to successfully perform the systems engineering required to develop an executable design.
Biography

Mr. James LONG, USA, is the Chief Methodologist and former president of Vitech Corporation. He has been a performing system engineer and innovator since creating the first behavior diagrams (then called Function Sequence Diagrams) at TRW in 1967. He played a key technical and management role in the maturing and application of that system engineering process and technology at TRW and Vitech. Mr. Long worked on many system developments with an emphasis on complex MIL/AERO, satellite, and C3I systems with embedded software for over 50 years. He is a member of INCOSE, active in NDIA’s Systems Engineering Division and its M&S Committee, and supported the OMG’s efforts to expand UML 2.0 to systems engineering (SysML).

Mr. Long has authored many technical papers and delivered tutorials in system engineering techniques and applications to much of the Defense and Intelligence community. Mr. Long received the M.S. in Astronautics from Purdue and the B.S. in Mechanical Engineering from General Motors Institute.

Mr. Long has been elected as a Fellow of INCOSE and was also selected as an Eminent Engineer by Tau Beta Pi, the honorary engineering scholastic society. The eminent engineering designation is recognition for career achievement in engineering.
Model-Based Systems Engineering For Project Success: The Complete Process

James E. Long
Chief Methodologist

David Long
President

Vitech Corporation
July 2010
• A brief introduction to Model-Based Systems Engineering (MBSE)
• Applying a MBSE process
• Service Oriented Architectures (SOAs)
• Overview of SE and DoDAF 2.0
• MBSE in practice: Developing a system of systems
• Summary and review
A Brief Introduction to Model-Based Systems Engineering
Systems Engineering: A Practice in Transition

Past

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

Future

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

Presented at INCOSE 2010 Symposium
Systems Engineering: Broad Applicability

- Reduce production COST and DESIGN-CYCLE TIME
  - Project: Advanced Cat Scan
  - Customer: Medrad

- Mitigate RISK during complex SOFTWARE INTEGRATION efforts
  - Project: Overnight Delivery
  - Customer: FedEx

- Improving PROCESSES to manage COST of PRODUCTION
  - Project: Diamond Exploration and Recovery
  - Customer: De Beers

- Ensure design CONSISTENCY with integrated DISCRETE-EVENT SIMULATION
  - Project: ICBM
  - Customer: Northrop Grumman for USAF

- Model and SIMULATE complex system LOGIC and BEHAVIOR
  - Project: Elevator Assembly
  - Customer: Otis Elevator

- Building systems from the ground up
  - Project: Satellite
  - Customer: Orbital Sciences
The Hidden Complexity of Systems Engineering

Constraints
Verification
Performance Issues
Trade Studies
MOE’s

Documents

Traceability
Decisions
Validation
Reviews
Interfaces

Risks
Architecture
Change Management

Requirements
Components
Model-Based Systems Engineering

- Formalizes the practice of systems engineering 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

Life Cycle Support

Concept → System Development → Production → Operations & Support

Vertical Integration

Reprinted from INCOSE Model-Based Systems Engineering Workshop, February 2010
Setting the Context – The Four Primary Systems Engineering Activities

Source Requirements Domain

Behavior Domain

Behavior is allocated to physical components

V&V Domain

Architecture Domain

Verified by

Originating requirements trace to physical components

Verified by

Verified by

Presented at INCOSE 2010 Symposium
Primary Design Traceability; It’s Done with Relationships (Verbs), Not Attributes (Adjectives)

Traceability:
- The parts of the system design that satisfy specific originating requirements
- The decision history
- The basis for subsequent changes in originating requirements

Document refined by Documents

Requirement basis of Requirement

Issue refined by Issue

Function allocated to Function

Component built from Component

Presented at INCOSE 2010 Symposium
Common SE “Tool Suite” Architecture

Requirements Management

Behavioral Analysis

Architecture Synthesis

Verification

Multiple products utilizing independent databases forces extraordinary data management – and complicates the original SE effort

Presented at INCOSE 2010 Symposium
Integrated, Consistent Analysis: Diagrams, complete specifications, and project work products automatically generated from the integrated model
The Systems Engineer’s Dilemma: Integration and Synchronization

Source Documents

- extracted requirements
- function list
- data items
- traceability list
- physical components
- interface definitions
- analyses & trade studies
- open action items
- engineering notebook

Printed Reports, Models, & Specifications

Any change will affect something else

Systems Engineer’s Desktop

Source Documents

- extracted requirements
- function list
- data items
- traceability list
- physical components
- interface definitions
- analyses & trade studies
- open action items
- engineering notebook

Printed Reports, Models, & Specifications

Any change will affect something else

Systems Engineer’s Desktop
Why are there Problems with SE as Commonly Practiced Today?

- Outdated approaches (document-based SE and viewgraph engineering)
- Other detailed issues
  - Underestimating the complexity
  - Failure to develop and manage the proper set of requirements
  - Failure to understand operational concepts
  - Too much reliance on a few experienced people
  - No repeatable process (CMM)
  - Information holes
- Inadequate tools to help with the entire process
  - Most tools help in specific areas (e.g. software development, design [CAD], etc.)
- Increasing use of COTS systems and components
- Shift toward architectures and Systems of Systems (SoS)
Essential Components of MBSE

- MBSE language (encompass at least problem, solution, and management domains)
  - Graphical control constructs
  - Behavior
  - The repository
- Model-Based Systems Engineering process
- Automatic generation of key documentation, design artifacts, and other work products
The system engineering process needs to support top-down, middle-out, and reverse engineering approaches to system specification and design.
Features of a Complete Systems Engineering Process

• Convergent
• Model-based
• Concurrent engineering
• Layered, hierarchical solution
• Central engineering repository
  – Incorporating system definition language
    • With graphical control constructs
    • Executable, dynamic validation of system logic
    • Context free
• Different initial conditions
  – Top-down
  – Middle-out
  – Reverse engineering
• Accommodates COTS, GOTS, …. concepts
• Automated artifact generation
What is a Model?

• A model is a limited representation of a system or process. Its role is to answer questions about the entity it represents.

• Types of models include
  – Executable
  – Information
  – Design
  – Operations
  – Process
  – Enterprise
  – Organization

• Models can be migrated into a cohesive unambiguous representation of a system.
MBSE – Three **Synchronized** Models are Necessary and Sufficient to Completely Specify a System

1. Control (functional behavior) model

2. Interface (data I/O) model

3. Physical architecture (component) model

What about performance requirements / resources?
- Captured with parts/combinations of the above models

Models provide basis for knowing when you are done. Selection of views is important; some provide more insight than others.
Why is a System Definition Language (SDL) Needed?

• Putting systems engineering information in a database is like dumping data into a bucket. Without structure and semantics it means little.

• SDL provides a structured, common, explicit, context-free language for technical communication.

• SDL serves as a guide for requirements analysts, system designers, and developers.

• SDL provides a structure for the graphic view generators, report generator scripts, and consistency checkers.
Impacts of Model-Based Systems Engineering

- Systems engineering paradigm shift
  - vs. text-based or diagram-based
- System model is essential and required
- System model encompasses the system design, execution, and specification
- System specifications are complete and consistent
- Model is provided to subsequent engineering teams
- Provides process for generation of complete, consistent, executable system design and specification

The MBSE technology empowers engineering teams to build a complete and integrated system definition.
The MBSE System Definition Language

SDL is an Extended Natural Language in ERA Format (Early Object-Oriented Language for Systems and Models)

<table>
<thead>
<tr>
<th>SDL Language</th>
<th>English Equivalent</th>
<th>MBSE Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Element</td>
<td>Noun</td>
<td>• <strong>Requirement:</strong> Place Orders</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <strong>Function:</strong> Cook Burgers</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <strong>Component:</strong> Cooks</td>
</tr>
<tr>
<td>Relationship</td>
<td>Verb</td>
<td>• <strong>Requirement basis of Functions</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <strong>Functions are allocated to Components</strong></td>
</tr>
<tr>
<td>Attribute</td>
<td>Adjective</td>
<td>• Creator</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Creation Date</td>
</tr>
<tr>
<td>Attribute of Relationship</td>
<td>Adverb</td>
<td>Resource consumed by Function</td>
</tr>
<tr>
<td>Structure</td>
<td>N/A</td>
<td>• Viewed as Enhanced FFBD or Activity Diagram</td>
</tr>
</tbody>
</table>
A Set of Complete and Executable Graphical Constructs (Structured Representations)

**SEQUENCE**

**SELECT**

**MULTIPLE-EXIT**

**ITERATE**

**LOOP**

**CONCURRENCY**

Presented at INCOSE 2010 Symposium
A Set of Complete and Executable Graphical Constructs (SysML Representations)

- **SEQUENCE**
- **SELECT**
- **MULTIPLE-EXIT**
- **ITERATE**
- **LOOP**
- **REPLICATE**
- **CONCURRENCY**
The Power of Models and Graphical Representations

• Communication between people with differing specialties and backgrounds
  – Universal language
  – Very powerful
  – Essential
  – Not context sensitive

• Modeling language for architectures
  – Physical
  – Functional

• Language to support
  – Requirements capture
  – System boundary definition
  – Threads, operational architecture, and system architecture development
  – Traceability
  – Impact analysis
  – Dynamic verification
  – Automatic documentation
Integrating the Four Primary SE Activities through a Design Repository

Utilizing a layered approach to progressively clarify and elaborate all four domains concurrently ensures consistency and completeness.
Integrating the Repository and View Generators Provides Consistency
View Generators using a Common Repository Guarantee Consistent Views

• A graphical view is defined by features and a format
• The features are in the repository
• The format for each view type is defined in the view generator

Views are projections of the model. Choose the views that serve the purpose.
A Momentary Aside for Some Insight –
The Control Enablement & Data Triggering Spectrum

Behavior Characteristics Spectrum

More complex control - Less control complexity
Less data triggering - More data triggering

Combination of:
- Control
- Control constructs
- Data triggering
- Data stores
- Completion criterion

All control
Control constructs
No data
No data triggering

All data
Data triggering
Data stores
No control constructs

Presented at INCOSE 2010 Symposium Slide 29
Relationships of the Graphical Representations - FFBDs & DFDs are Limiting Cases

Dynamic Timelines

EFFBD / Behavior Diagrams
- Provide Both Data Triggering and Control Constructs
- Balance Depends on Needs and Analyst
- Diagram is Executable

EFFBD (CORE)

BD (DCDS)

Data Flow Diagram
- Only Data Triggering
- No Control Constructs

N2 Chart
- Equivalent to DFD

Behavior Characteristics Spectrum

- More complex control
- Less data triggering

- Less control complexity
- More data triggering

Function Flow Block Diagram
- Only Control Constructs
- No Data Triggering

IDEF0 Diagram
- Primarily DFD
- Some Control, No Control Constructs

Use Case
- Equivalent to DFD

Sequence Diagram
- Message Flows

Presented at INCOSE 2010 Symposium
Applying an MBSE Process
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.
Model-Based System Engineering 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 Reqts
8. Update System Boundary
7a. Modify Reqts & Arch. Constraints
6a. Modify System Threads
5a. Modify & Decompose System Behavior
4a. Allocate Behavior to Components
3a. Refine Component Hierarchy
2a. Define Interfaces
9. Select Design
10. Perform Effectiveness & Feasibility Analyses
12. Develop Test Plans
13. Generate Documentation and Specifications

Find the top, then modify top-down.

SCHEDULE

Presented at INCOSE 2010 Symposium
MBSE – the Onion Model

Doing Systems Engineering in Increments/Layers

Primary Concurrent Engineering Activities At Each Layer

Layer 1
(Draft 1)

Layer 2
(Draft 2)

Layer n
(Final Specs.)

Must complete a layer before moving to the next layer (completeness)
Cannot iterate back more than one layer (convergence)
Concurrent Engineering Enables “Design It In”
Don’t Try to Test It In, Review It In, Annotate It In …

“It”:
Complete
Consistent
Correct
Implementable
Current
…….

System Design Repository

Program Mgt.
Configuration Management
Customer
Chief Engineer
Hardware
Software
Safety
Reliability, Availability, Maintainability
Manufacturability
Security
Test
Logistics
Maintenance
Operations
Environmental
Training & Personnel
Publications

Outer ring represents domain experts

Presented at INCOSE 2010 Symposium
Slide 35
The Image Management System (IMS) Overview

- Customers
- Customer Link
- Image Management System
- Image Collector Link
- Image Collectors

Presented at INCOSE 2010 Symposium
Essential Tasks Before You Start Development Activities

- Plan the activity.
  - Prepare a Systems Engineering Management Plan (e.g., SEMP).
  - Tailor the plan to your project.

- Make sure you assign responsibility.
  - Define the (group of) people who retain authority over the system requirements, behavior, architecture, interfaces, and test and integration plan.
A Process for Engineering the Image Management System

- Define the system
- Capture originating requirements
  - Evaluate a source document
  - Extract requirements
- Define the system boundary
- Define the system behavior and physical architecture
- Allocate the behavior to the physical components
Capture the Originating Requirements
“Making Sure That We Solve the Right Problem”

- Start by extracting the first-level requirements from the source document(s) in order to gain an understanding of the top-level context of the system.
- Next capture the “children” of the first-level requirements, creating Issues as required.
- The objective is to continue the hierarchy of extracting “children” until each leaf-level requirement is a single, verifiable statement.
Candidate Source Documents

- 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
Capturing the System Requirements

1. The system shall accept information requests from certified customers. The system shall retain an inventory of previously collected images/products and provide them to users.

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

3. The system shall be staffed at a maximum of 35 personnel on any shift.

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

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

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

7. The system shall monitor its own performance.

8. The system shall assess its own performance.
The Requirements Capture Approach

- Get the leaf nodes – the requirements in single, verifiable statements.
- Record source requirement statement in the Description attribute of a **Requirement**.
- Obtain traceability between source and first level **Requirements** with documents/documented by relationships.
- Obtain traceability between parent/child **Requirements** with the refines/refined by relationships.
Issues

• During the requirements capture and analysis process, it is likely that problems will be found
  – Requirements are not clear or complete
  – Requirements may contradict each other
  – Requirements may be over or under specified

• It is highly desirable to have a mechanism to capture these issues, as well as the subsequent resolution of the issue and supporting documentation

• This is accomplished using **Issue** elements
Risk

• Defined as the uncertainly of attaining or achieving a product or program milestone

• May exist for several reasons
  – Budget or schedule constraints
  – High or leading edge technology
  – New designs or design concepts
  – Criticality to the user/customer

• We capture this information by using the Risk element
The System Physical Boundary

- Referred to as the system “Physical Context”
- Defines all external systems to which our system must interface and the mechanism(s) for interfacing
- Provides a structure into which behavior can be partitioned and data assigned to interfaces

Image Management System

Customers

Collectors

Physical Context

Puts a boundary on the system problem so we don’t add something that is not intended or needed

Presented at INCOSE 2010 Symposium
System 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

Behavior is essential for providing the complete systems engineering model of any system or process.
The Many Faces of Behavior

EFFBD or Activity Diagram
Complete and Executable

FFBD Lacks data

Sequence Diagram Lacks structure

N2 Diagram Lacks constructs

Property Sheet

IDEF0 Lacks constructs

Presented at INCOSE 2010 Symposium
Identifying Use Cases

- Derived from system context, operational concept, and requirements

- Should include
  - Preconditions
  - Primary flow
  - Alternate flow(s)
  - Post-conditions

- Elaborated by system threads or system behavior
Use Case Relationships

- Use cases use relationships to describe their place and role in the system
  - **Communication**: the external “actors” interact with the system through communication
  - **Include**: this relationship allows use cases to reuse functionality from the base use cases
  - **Extend**: under certain specified conditions one use case may extend the functionality of another
  - **Generalization**: allows for the description of variants on a base use case
Threads Offer Insight Into How the System Must Respond to Its Stimuli

- Definition: A thread is a single stimulus/response behavior path through a system.
- Threads give us insight on what the system must do (functions) to respond to different classes of system stimuli to produce desired outputs and behavior.
- Thread derivation is a discovery process.
- Thread derivation also validates completeness of the functional source requirements.

Activity Diagram
From Threads to Integrated Behavior or Operational Architecture

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

Presented at INCOSE 2010 Symposium

1. Derive Threads

2. Integrate Threads to Define Integrated System Behavior
Conditions for a Function to Execute

- **Execution**
  - Before a Function can begin execution it must be enabled and triggered, if a trigger is defined

- **Enablement**
  - Enablement is a control concept
  - A Function is enabled upon completion of the Function prior to it in the construct

- **Triggers**
  - A Trigger is an Item that also provides a control role
  - Trigger is defined by a relationship (triggers)
  - Triggers are shown with a double arrowhead

- **Data Stores**
  - A Data Store is an Item that does not provide a control role
  - Data store is defined by a relationship (input to)
  - Data stores are shown with a single arrowhead
System’s behavior is described as a “black box” to identify conditional process flow and performance.

- Design is done by “allocation” of functions and performance onto the components, then interface design.
- Design is then analyzed by all “disciplines” and iterated.

Ref: DCDS Documentation
Relating the Functional and Physical Models

Functional Hierarchy

- Functional Context
  - Operate Image Management System
    - Function A
    - Function B
    - Function C
    - Function D
    - Function E
    - Function F
    - Function G
    - Function H
    - Function I
    - Function J
    - Function K
    - Function L
    - Function M
    - Function N
      - Perform Customer Functions
      - Perform Collector Functions

Physical Hierarchy

- Physical Context
  - Image Management System
    - Workstation
    - Command Center
  - Customers
  - Collectors

- Physical Context
  - allocated to (performs)
  - decomposed by
  - allocated to (performs)
  - built from
Decomposition – The Problem

- We have stated that decomposition is the reverse of aggregation. We have not defined what properties must be preserved under decomposition.
- How do we go from the top down? i.e., what must be true to say that a function is decomposed by a graph of functions?
- Is decomposition unique?
  - Remember we use the black box approach
Why Aggregate?

Most engineers start with too much detail

• Enhance understanding
  – See the “big picture”
  – Simplify the look of a graph

• Encapsulate complex logic sequences into larger building blocks
  – Hide information
Allocation of Behavior to the System’s Internal Components

• Inside the system boundary, the deliverable system consists of a **collection of cooperating component parts** with a common goal.

• The allocation process partitions the system-level conceptual behavior among the system’s internal component parts.

• Must preserve the specified system behavior during the allocation process (functional/performance behavior).

• Perform trial allocations to determine the best partitioning such that
  – Behavior is preserved, and
  – Interfaces are not too complex to build

Ref.: DCDS Documentation
Allocation Implications

• Moves the design
  – From abstract to concrete
  – From logical to top-level physical
• Creates basis for interface design
• Precipitates consideration of physical implications:
  – Physical limitations (e.g., weight, size, heat)
  – Resource constraints
  – Failure detection and recovery
  – Manufacturability
  – Maintainability

Ref.: DCDS Documentation
Sequence diagrams focus on triggering information between activities on lifelines to help you understand the way the activity interacts with the greater system.
Allocating the System Functions

The Components will perform the Functions that are allocated to them.
Capturing the Interfaces

Interfaces:
- Logical
- Bi-directional

Links:
- Physical
- Have capacity

Items:
- Have size

We have established the **Items** that cross the **Interfaces** by allocating **Functions** to Components.
Failure Modes and Effects Analysis

For each component and interface determine:

• Requirements and performance indices

• Potential failure modes
  – Probability of each
  – Avoidance approaches
  – Detection strategies
  – Recovery strategies
    • Feasible to continue in spite of failure?
    • Feasible to resume normal behavior after replacement, repair, or end of intermittent failure?

• Impact of avoidance and/or detection and recovery
  – In high reliability systems this may be the majority of the system behavior

Ref.: DCDS Documentation
The Discrete Event Simulator Supports Analysis and Design at All Levels

- **Dynamic analysis**
  - Assesses dynamic consistency/ executability of “system behavior”

- **Timeline analysis**
  - Establishes and analyzes integrated behavior timelines

- **Resource analysis**
  - Monitors the amounts and dynamics of system resources: e.g., People, computer MIPS, memory, supplies, power, radar pulses, # of interceptors, # of cooked hamburgers, ...

- **Flow analysis**
  - What happens to system operation when items of finite size are carried by links of finite capacity?
The Executed Behavior Confirms System Logic and Supports Trade Studies

Simulator for Context Function - Level 2 (IMS Project)

166.14

C. 1.1 Make Information Request
C. 1.2 Receive Estimated Schedule
C. 1.3 Accept Products
1. Accept And Format Request
2. Check Product Inventory
3. Prioritize Request
4. Determine Collector Mix
5. Notify User Of Estimated Schedule
6. Task Collectors
7. Accept And Format Collector Products
8. Put Product In Inventory
9. Get Product From Inventory
10. Provide Product To User
11. Evaluate Products vs. Request
12. Report Deficiencies And Recommendations...
C. 2.1 Collect Data
C. 2.2 Process and Provide Collector Data

Transcript:
166. 364447, 364447, PROC(C, 2, 1), desqeq, 1.2.5, 2.2.1, 1.1, 1.2, "Report Deficiencies And Recommendations", "deficiency report"
166. 364447, 364447, PROC(C, 2, 1), start, 1.2.5, 2.2.1, 1.1, 1.2, "Report Deficiencies And Recommendations"
166. 14024, ID(58), PROC(C, 2, 1), finish, 1.2.5, 2.2.1, 1.1, 1.2, "Report Deficiencies And Recommendations"
166. 14024, ID(55), PROC(C, 2, 1), finish, 1.2.5, 2.2.1, 1.1, 1.2, "Report Deficiencies And Recommendations"
166. 14024, ID(55), PROC(C, 2, 1), finish, 1.2.5, 2.2.1, 1.1, 1.2, "Report Deficiencies And Recommendations"
166. 14024, ID(55), PROC(C, 2, 1), finish, 1.2.5, Parallel
166. 14024, ID(55), PROC(C, 2, 1), finish, 1.2.5, Parallel
Service Oriented Architecture (SOA)
Pre-SOA Configuration
SOA Basics

Process (Orchestration) Layer
SOA Basics

Services Layer
Unified Views

Metrics Points
Power of MBSE for SOA

- Process drives the architecture
- Services are derived coherently
- Metrics are meaningful
- Views serve the functional roles
DoDAF 2.0 and a MBSE Roadmap for Generating Architectures
What is an Architecture?

An architecture is the fundamental organization of a system embodied in its components, their relationships to each other, and to the environment and the principles guiding its design and evolution.

IEEE STD 1472 (2000)

Architectures are a primary tool for enterprise-level systems integration.


[Architecture] is the set of design decisions which, if made incorrectly, may cause your project to be canceled.

Eoin Woods
Architectures in Context


Presented at INCOSE 2010 Symposium
Integrated Architecture

- An architecture where architecture data elements are uniquely identified and consistently used across all products and views within the architecture.

Federated Architecture

- Provides a framework for enterprise architecture development, maintenance, and use that aligns, locates, and links disparate architectures via information exchange standards (i.e., taxonomies).

EA Reference Model Taxonomy

- Managed by MA Authority

Subordinate architectures mapped to MA-level by C/S/As

Presented at INCOSE 2010 Symposium
From Architectures to a Framework: Why is a Framework Needed?

Organizations are developing major systems that need to interface and interact.

Differences in content and formats inhibit comparison of architectures.

Disparate and unrelatable architecture products lead to non-integrated, non-interoperable, and non-cost effective capabilities in the field.

DoDAF Evolution To Support “Fit For Purpose” Architecture

DoDAF 1.0
- CADM Separate
- Baseline For DoDAF 1.5
- Removed Essential & Supporting Designations
- Expanded audience to all of DoD

DoDAF 1.5
- Addresses Net-Centricity
- Volume III is CADM & Architecture Data Strategy
- Addresses Architecture Federation
- Baseline for DoDAF 2.0
- Shifted away from DoDAF mandating a set of products

DoDAF 2.0
- Cover Enterprise and Program Architecture
- Emphasize Data versus Products
- Tailored Presentation
- DM2 PES

Presented at INCOSE 2010 Symposium
DoDAF 2.0: A Marked Expansion

DoDAF V2.0

DoDAF V1.5

All "Service" Versions of SV Products

All “Systems” Versions of SV Products

Systems View

Services View

Data & Information View

Operational View

System Engineering View

Capability View

Project View

Dashboards

Graphical Depictions

Reference Models

Fusion Products

Composite Products

Fit For Purpose Presentation

All View

Standards View

Operational View

Rest of OVs

All AVs

All TVs

Updated

Moved

New

DoDAF Metamodel (DM2)

Presented at INCOSE 2010 Symposium
DoDAF 2.0: A Marked Departure

• Movement from a product-centric approach to a data-centric approach
  – Provide decision-making data organized as information for the manager/executive

• Architecture development as a management tool
  – Support the decision-making process of the executive as process owner
  – Ensure a particular process or program
    • Works efficiently
    • Complies with legal and departmental requirements
    • Serves the purpose for which it was created

• Viewpoint selection by the process-owner based upon “fit-for-purpose”
  – Choose the viewpoints that accomplish the objective
A Data-Centric Approach Supporting “Fit for Purpose” Views
DoDAF 2.0 Model, View, and Viewpoint Concepts

- **Model** – a template for collecting data
- **View** – a representation of a related set of information using formats or models
  - Dashboards
  - Spreadsheets
  - Diagrams
  - Data models
  - Any presentation style that conveys the meaning of the data

The architect translates the decision-maker’s requirements into a set of data that can be used by engineers to design possible solutions.
DoDAF 2.0 Model, View, and Viewpoint Concepts, cont.

• **Viewpoint** – one or more organizing perspectives for data useful for supporting management decision-making, including
  – The information appearing in individual views
  – How to construct and use the views (by means of an appropriate schema or template)
  – The modeling techniques for expressing and analyzing the information
  – A rationale for these choices (e.g., by describing the purpose and intended audience of the view)

Architecture Description:
• a collection of products to document an architecture (ISO 42010)
• a collection of views to document an architecture (DoDAF 2.0)
DoDAF 2.0 Viewpoints

**Capability Viewpoint**
Articulate the capability requirement, delivery timing, and deployed capability

**Operational Viewpoint**
Articulate operational scenarios, processes, activities & requirements

**Services Viewpoint**
Articulate the performers, activities, services, and their exchanges providing for, or supporting, DoD functions

**Systems Viewpoint**
Articulate the legacy systems or independent systems, their composition, interconnectivity, and context providing for, or supporting, DoD functions

**Project Viewpoint**
Describes the relationships between operational and capability requirements and the various projects being implemented; Details dependencies between capability management and the Defense Acquisition System process.

**Standards Viewpoint**
Articulate applicable Operational, Business, Technical, and Industry policy, standards, guidance, constraints, and forecasts

**Data and Information Viewpoint**
Articulate the data relationships and alignment structures in the architecture content

**Overarching aspects of architecture context that relate to all views**

Presented at INCOSE 2010 Symposium
Envisioning Architecture Scope

DoD Decision-Making Activities
- Planning
- Requirements, JCIDS
- PPBE
- DAS
- Warfighter and other users

Direction, Guidance Impact, Results
- Strategic Plans
- GIG Arch Vision
- NC Data Strategy
- NC Services Strategy
- NC IA Strategy

Joint Ops Concepts CONOPS
- DOTMLPF changes
- JDF/Planning Guidance
- Portfolios, Increments
- POM
- Programs
- Operational Systems

Mission Effectiveness

Architecture & Engineering Scope and Focus
- Department Level
  - Capability Level (supports PIM)
    - 8115
    - CPMs
    - ICPs
- Systems Architecting (supports PEOs/PMs)
- Mission Operations and Support

Enterprise Architecture

Enterprise-wide System Engineering

Presented at INCOSE 2010 Symposium
Communication Mismatch: Manager, Architect, and Domain Experts

Presented at INCOSE 2010 Symposium
Challenges in Satisfying the Intent of DoDAF

• Delivery of DoDAF views does not guarantee a complete, consistent solution
  – A complete set of DoDAF views does not necessarily meet the goals of DoDAF (particularly prior to DoDAF 2.0)
  – Ambiguities remain from original product focus
  – A critical original intent (comparisons of architecture cost, schedule, and traceability) remains unfulfilled

• Confusion surrounding architecture development
  – Views must be consistent

• False assumptions (ex., UML is the standard for DoDAF)

• Continued evolution of DoDAF
  – From product focus to data-centric
  – Interest in integrated, executable architectures growing
Solution: Make Your DoDAF Efforts Part of the Greater Architecture / SE Effort

- Leverage a defined systems engineering process with quality automated tools
  - Integrated schema/language/repository
- Model the operations domain as well as the systems domain
- Generate DoDAF views as intentional byproducts of the architecture / systems engineering effort
  - Reinforced by “fit for purpose” direction of DoDAF 2.0
- Maintain traceability of integrated operations modeling to system engineering modeling to DoDAF views
Implementation of integrated DoDAF and system engineering processes provides:

- A repeatable and proven model-based systems engineering methodology
- Integrated models for:
  - technical,
  - operational, and
  - system architectures
- A graphical notation to enhance capture and representation for:
  - communication and
  - evaluation of candidate architectures
- Executable models (simulation) for behavioral and performance analysis
- Consistent DoDAF products produced directly from the system design repository:
  - Support for the product life cycle
  - Significant savings in cost and schedule
DoDAF 2.0 changes denoted in bold red

Presented at INCOSE 2010 Symposium
DoDAF 2.0 Schema: Capturing the Program Dimension

Program Management Domain

Program Activity

Selected Classes

Product

System Architecture Domain

Capability refined by

Requirement refined by

Operational Activity

Organizational Activity

Operational Task

Mission

Organization

Selected Classes

Operational Architecture Domain

Program Element

Risk

Architecture

Exit

Resource

Component

Recorded at INCOSE 2010 Symposium Slide 96
Relationship between OV and SV Generation

Operational Analysis/Operational View

- Provides top level Functional Requirements as output
- Individual or groups of requirements may be allocated to different systems (SOS)
- One, two layers tops

Systems Design/Systems View

- “Implemented By” relationships between lowest level of OV’s and top level of SV’s

Presented at INCOSE 2010 Symposium
Model-Based Systems Engineering Activities Timeline – Top Down

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

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

SCHEDULE

Presented at INCOSE 2010 Symposium
Traditional DoDAF Views within the Systems Development Timeline

1. Capture and Analyze Related Documents
2. Identify Assumptions
3. Identify Existing / Planned Systems
4. Capture Constraints
5. Develop Operational Context Diagram
6. Develop Operational Scenarios
7. Derive Functional Behavior
8. Derive System Elements
9. Allocate Functions to System Elements
10. Prepare I/F Diagrams
11. Define Resources, Error Detect & Recovery
12. Perform Dynamic Analysis
13. Develop Operational Demonstration Master Plan
14. Provide Options
15. Conduct Trade-off Analyses
16. Generate Operational and System Architecture Views, Briefings, and Reports

Color Code:
- Synthesis
- System Analysis & Control
- Behavior Analysis
- Requirements Analysis

Presented at INCOSE 2010 Symposium
MBSE in Practice: 
Developing a 
System of Systems 

Source documents and graphics from publicly available sources where indicated
What is a System?

• A system can be broadly defined as an integrated set of elements that accomplish a defined objective. (INCOSE SE Handbook - 2004)

• An integrated composite of people, products, and processes that provide a capability to satisfy a stated need or objective. (EIA/IS-632 - 1994)

• A set or arrangement of elements (people, products (hardware and software) and processes (facilities, equipment, material, and procedures) that are related and whose behavior satisfies customer/operational needs, and provides for the life cycle sustainment of the products. (IEEE-1220-1998)
What is a System of Systems (SoS)?

• A System of Systems is a system in which:
  – System elements are predominantly systems in their own right
  – Individual operational threads involve multiple system elements
  – A SoS view emphasizes interoperability among the elements
  – A SoS will likely include systems from different families
    • e.g., SoS combining C4I and weapon systems
  – The SoS architect defines the SoS structure, but may not control the implementation of all system elements
Multiple Classes of SoS

• New: Systems comprising SoS do not currently exist
  – Use a traditional spiral or vee approach

• Integration: SoS integrates multiple existing systems
  – Use a reverse engineering approach

• Hybrid: SoS integrates a mix of existing and new systems
  – Use a top-down/bottom-up/middle-out systems engineering approach
Challenges Posed by SoS

• Insufficient effort and rigor in the requirements and analysis phase given expanded scope
• Programmatic issues resulting from multiple user, acquisition, and implementation teams
• Lack of centralized control over budgets and resources
• Technical complexity at the interfaces
• Independently evolving system elements resulting in shifting interfaces during the lifecycle of the SoS
A Definition of System to Keep in Mind for SoS

a whole that cannot be divided into independent parts without losing its essential characteristics as a whole.

It follows from this definition that, a system’s essential defining properties are the product of the interactions of its parts, not the actions of the parts considered separately.

Therefore, when a system is taken apart, or its parts are considered independently of each other, the system loses its essential properties.

Furthermore, when performance of each part taken separately is improved, the performance of the system as a whole may not be, and usually isn’t.

--Russell Ackoff
Stakeholder Requirements for the Missile Defense System of Systems

- Protect the United States against limited ballistic missile threats, including accidental or unauthorized launches or Third World threats.
- The means to accomplish the mission are as follows:
  - Detect the launch of enemy ballistic missile(s) and track.
  - Continue tracking of ballistic missile(s) using ground based radars.
  - Engage and destroy the ballistic missile warhead above the earth’s atmosphere by force of impact.

MBSE – The Onion Model for the SoS Example

Primary Concurrent Engineering Activities At Each Layer

<table>
<thead>
<tr>
<th>Layer</th>
<th>Originating Requirements Analysis</th>
<th>Behavior Analysis</th>
<th>Synthesis/Architecture</th>
<th>Design V &amp; V</th>
</tr>
</thead>
<tbody>
<tr>
<td>Layer 1 (Draft 1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Layer 2 (Draft 2)</td>
<td>Initial Requirements for this layer are embodied in the model passed from the prior layer</td>
<td>Behavior Analysis</td>
<td>Synthesis/Architecture</td>
<td>Design V &amp; V</td>
</tr>
<tr>
<td>Layer n (Final Specs.)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

System Design Repository

- Iterate as required
- When layer completed

Specification & Report Generation

- Iterate as required
- When layer completed

Must complete a layer before moving to the next layer (completeness)
Cannot iterate back more than one layer (convergence)

Presented at INCOSE 2010 Symposium
Overview of the MBSE Process for this Hybrid SoS

• Level 1: Define the SoS Mission
  – Capture stakeholder requirements
  – Develop SoS scenarios to discover required SoS basic functions
  – Identify the candidate SoS operational elements/systems
  – Define to-be interfaces

• Level 2: Continue definition of SoS and System Design Repository
  – Integrate scenarios to determine desired SoS to-be functional architecture
  – Capture the as-is system functions and interfaces resulting from the candidate SoS elements/systems
  – Identify the differences between the to-be SoS behavior and interfaces and the aggregation of the existing SoS as-is elements

• Level 3: Design to-be SoS elements and modify/augment current SoS elements to achieve SoS required capabilities
LEVEL 1 of the Onion Model for this SoS Example

Primary Concurrent Engineering Activities At Each Layer

- Originating Requirements Analysis
- Behavior Analysis
- Synthesis/Architecture
- Design V & V

Top-level Reqts.

- System Design Repository
- Specification & Report Generation

Next-level Reqts.

- System Design Repository
- Specification & Report Generation

Layer 1
(Draft 1)

Layer 2
(Draft 2)

Layer n
(Final Specs.)

Level 1: Define the SoS Mission
- Capture stakeholder requirements
- Develop SoS scenarios to discover required SoS basic functions
- Identify the candidate SoS operational elements/systems
- Define to-be interfaces

Must complete a layer before moving to the next layer (completeness)
Cannot iterate back more than one layer (convergence)
SoS Source Document and Requirements are Extracted and Shown in Traceability Hierarchy

# Details of the SoS Source Requirements, Showing Relationships and Attributes

<table>
<thead>
<tr>
<th>Number &amp; Name</th>
<th>Description</th>
<th>documented by</th>
<th>incorporated in</th>
<th>incorporates</th>
</tr>
</thead>
<tbody>
<tr>
<td>OR.1 Provide ABM defense for the US</td>
<td>Protect the United States against limited ballistic missile threats, including accidental or unauthorized launches or Third World threats. The means to accomplish the NMD mission are: (1) Detect the launch of enemy ballistic missile(s) and track. (2) Continue tracking of ballistic missile(s) using ground based radars. (3) Engage and destroy the ballistic missile warhead above the earth’s atmosphere by force of impact.</td>
<td>SD.1 NMD Description Document by Federation of American Scientists (FAS)</td>
<td></td>
<td>OR.1.1 Detect and track OR.1.2 Track with GBR OR.1.3 Engage and destroy warhead</td>
</tr>
<tr>
<td>OR.1.1 Detect and track</td>
<td>Detect the launch of enemy ballistic missile(s) and track.</td>
<td></td>
<td>OR.1 Provide ABM defense for the US</td>
<td></td>
</tr>
<tr>
<td>OR.1.2 Track with GBR</td>
<td>Continue tracking of ballistic missile(s) using ground based radars.</td>
<td></td>
<td>OR.1 Provide ABM defense for the US</td>
<td></td>
</tr>
<tr>
<td>OR.1.3 Engage and destroy warhead</td>
<td>Engage and destroy the ballistic missile warhead above the earth’s atmosphere by force of impact.</td>
<td></td>
<td>OR.1 Provide ABM defense for the US</td>
<td></td>
</tr>
</tbody>
</table>

Presented at INCOSE 2010 Symposium
Operational Scenario Provides the Basis for Defining the Top Level System Functions

Source of graphic: Boeing website, www.boeing.com
Scenario 1: Provide Basic SoS Functions

- Detect Object
- Track Object
- Discriminate Object
- Plan Intercept
- Engage Object
- Performa Kill Assessment
Make-or-Buy Decisions

• During system design/development, always look for implementation alternatives

• Therefore, at each level of decomposition, look for existing implementation to satisfy needs
  – If system element is directed by stakeholder requirement, design to incorporate it
  – If a satisfactory implementation is available, design to incorporate it
  – Otherwise, complete design at this level, and

• Continue to the next level
Pre-existing System Elements: What Do You Get?

- Two features that are pre-defined
  - Physical interfaces
  - Behavior (stimulus-response characteristics)
- Restriction/constraint
  - Your system must be designed to match these features
- Future element releases/upgrades
  - That you may not influence
Considerations in Make-or-Buy Decisions

- Cost
- Interfaces
- Stimulus/response
- Agility
  - Command & control
  - Competing resources
  - Error detection and recovery
- Balance of elements: H/W, S/W, & humans
- System evolution
Candidate Elements (Systems) for the Missile Defense System

Built From

SoS Elements (Initially Stand-Alone Systems)

System of Systems

Source of graphics: Boeing website, www.boeing.com
LEVEL 2 of the Onion Model for this SoS Example

Primary Concurrent Engineering Activities At Each Layer

Layer 1 (Draft 1)
- System Design Repository
- Specification & Report Generation
  - Iterate as required
  - When layer completed

Layer 2 (Draft 2)
- System Design Repository
- Specification & Report Generation
  - Iterate as required
  - When layer completed

Level 2: Continue definition of SoS and System Design Repository
- Integrate scenarios to determine desired SoS to-be functional architecture
- Capture the as-is system functions and interfaces resulting from the candidate SoS elements/systems
- Identify the differences between the to-be SoS behavior and interfaces and the aggregation of the existing SoS as-is elements

Must complete a layer before moving to the next layer (completeness)
Cannot iterate back more than one layer (convergence)

Presented at INCOSE 2010 Symposium
Traceability Hierarchy Including the Selected SoS System Elements

NMD Description Document by Federation of American Scientists Document

OR.1
Provide ABM Defense for the U.S.

OR.1.1
Detect and Track Requirement

OR.1.2
Track With GBR Requirement

OR.1.3
Engage and Destroy Warheads Requirement

S.1
Space Based Infrared System (SBIR)

S.2
Upgraded Early Warning Radar (UEWR)

S.3
X-Band / Ground-Based Radars (XBR)

S.4
NMD Battle Management, Command and Control (BMC2)

S.5
NMD In-Flight Interceptor Communications System (IFICs)

OR.1.1
Detect and Track Requirement


Presented at INCOSE 2010 Symposium
N² Model Highlights Problems with Communication and Coordination Across the SoS System Elements

Absence of data flow/interfaces illustrates the need to integrate the top level elements/systems into a System of Systems.

Presented at INCOSE 2010 Symposium
Derived Physical Links are Shown via a Physical Block Diagram
EFFBD Shows SoS Top Level Functional Architecture and Interfaces

Presented at INCOSE 2010 Symposium
N2 Diagram Shows SoS Top-Level Interfaces, Independent of Time

1. NMC Management & Control
   - boost phase track status
   - object detection status

2. Object Detection and Boost Phase Track
   - midcourse track status
   - Object Discrimination and Terminal Track

3. Object Midcourse Track
   - interceptor status

4. Object Discrimination and Terminal Track
   - interceptor status

5. Interceptor Data
   - Kill Assessment
   - Terminal Track
   - XBR Discrimination

Presented at INCOSE 2010 Symposium
Additional Work to Complete Our SE Effort

- Capture as-is stimulus-response behavior for each of the candidate components
- Capture desired stimulus-response behavior for each of the components
- Analyze the differences and design the desired/necessary modifications to the current components
- Design tests to verify that augmented system will operate in an integrated mode
- Formulate a management process that will assure the existing components systems and the new desired component systems will operate as necessary
To Complete the Integration of the SoS Elements into an SoS System

• Continue to apply model-based system definition and development processes
  – Layered definition
  – Hierarchical decomposition
  – Executable architecture
  – Consistent design

• Continue to use a system definition repository and leverage graphical representations

• Continue until three models are sufficiently defined
  – Control
  – Input/Output
  – Physical architecture
• Challenges posed by SoS development are not fundamentally unique
  – SoS simply highlights the challenges posed by any complex development effort

• MBSE is a key enabler providing
  – Needed insight into the user requirements and solution space
  – Unambiguous executable architecture
  – Reduced programmatic risk

• Successful execution of an SoS program requires management commitment and understanding of the challenges
  – Technology alone is not the solution
Summary and Review
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.
Common SE Process Mistakes Today

• Process not convergent
• Functional model not distinct from Physical model
• Human and other physical components not included in single integrated model
  – Creates unnecessary, complex external interface
• Software broken out of integrated physical model too soon in the process
• Select implementation architecture too soon
• COTS not treated with enough care
  – Faster and cheaper today, at the expense of problems tomorrow
MBSE Benefits to the Enterprise

- MBSE supports the entire systems engineering process
- Its use clarifies the derivation and management of customer requirements
- The methodology provides a disciplined technical basis for informed decision making
- It supports identification and resolution of issues, interfaces, and risks
- It enables users to communicate and work in a team environment via a common repository
• Models allow simulation of the requirements:
  – Performance, inconsistencies, interface, throughput, resources

• Trade studies are substantiated with model outputs

• Life-cycle management of system models are traced to requirements:
  – System, subsystems, components, procurement, logistics, and deployment

• Documents are artifacts of the engineering process

• Improved system design and communication quality

• Enhanced risk tracking and identification

• Robust architecture V&V
Final Words

The Efficient SE Process
Essential Concepts/Benefits

• Language the problem and the solution space, include semantically-meaningful graphics to stay explicit and consistent
  – Traceability
  – Consistent graphics
  – Automatic documentation and artifacts
  – Dynamic validation and simulation
  – Easier and more precise communication
• Utilize a Model-Based Systems Engineering (MBSE) system design repository
• Engineer your system horizontally before vertically, i.e., do it in complete, converging layers
• Take advantage of the power of models
• Use tools to do the perspiration stuff. Use your brain to do the inspiration stuff
MBSE – Three **Synchronized** Models are Necessary and Sufficient to Completely Specify a System

1. Control (functional behavior) model

2. Interface (data I/O) model

3. Physical architecture (component) model

What about performance requirements / resources?

- Captured with parts/combinations of the above models

---

Models provide basis for knowing when you are done. Selection of views is important; some provide more insight than others.
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 using the Onion Model. Do Systems Engineering in Increments/Layers

**Primary Concurrent Engineering Activities At Each Layer**

- **Originating Requirements Analysis**
- **Behavior Analysis**
- **Synthesis/Architecture**
- **Design V & V**

**Layer 1**

- System Design Repository
- Specification & Report Generation
- Iterate as required
- When layer completed

**Layer 2**

- System Design Repository
- Specification & Report Generation
- Iterate as required
- When layer completed

**Layer n**

- System Design Repository
- Specification & Report Generation
- Iterate as required
- When layer completed

**Must complete a layer before moving to the next layer (completeness)**

**Cannot iterate back more than one layer (convergence)**

Presented at INCOSE 2010 Symposium

Slide 136
Don’t Waste Project Resources

• Designing to the three necessary and sufficient MBSE models keeps the activities focused and relevant

• Using the Onion model/layers keeps development areas synchronized.
  – Reduces breakage due to pre-emptive designs which must be re-done

• Using a good system design language, a repository, automatic graphic generation, executability, and automatic documentation provides maximum efficiency

• Do it right the first time!
Consistency is Essential (Quality) and Efficient (Cost, Maintenance, and Timeliness) – Use Improved Practices

<table>
<thead>
<tr>
<th>Viewgraph Engineering (Common Practice)</th>
<th>Model-Based SE (Improved Practice)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Independent drawings</td>
<td>Consistent views</td>
</tr>
<tr>
<td>Static diagrams</td>
<td>Executable behavior[1]</td>
</tr>
<tr>
<td>Data storage</td>
<td>Linked repository</td>
</tr>
<tr>
<td>Stored views</td>
<td>Dynamic view generation</td>
</tr>
<tr>
<td>Ad hoc process (inconsistent results)</td>
<td>Repeatable process (consistent results)</td>
</tr>
<tr>
<td>Manual change propagation across all affected products (by the systems engineer)</td>
<td>Automatic change propagation across all current and future products (by the engineering environment)</td>
</tr>
</tbody>
</table>

[1] Executable behavior eliminates structural or dynamic inconsistencies from the requirements.
How Do You Know When You Are Done (at Each Layer)?

<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 / Functional Architecture</td>
<td>2. Each function is uniquely allocated to at most one component</td>
</tr>
<tr>
<td>3. Physical Architecture Definition</td>
<td>3. Segment/component specs are complete 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>
How Do You Know When You Are Done with SE of the System?

• Within projected technology, you have an achievable design specification for all system components
• System V & V Plans are defined and fully traced
Thank You!

For additional information:

James E. Long
jlong@vitechcorp.com

David Long
dlong@vitechcorp.com

Vitech Corporation
2270 Kraft Drive, Suite 1600
Blacksburg, VA, 24060
1.540.951.3322
www.vitechcorp.com