Model-Based Systems Engineering For Project Success: The Complete Process

James Long, Vitech Corporation

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 DoDAF 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.

Model-Based Systems Engineering For Project Success: The Complete Process

James Long, Vitech Corporation

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

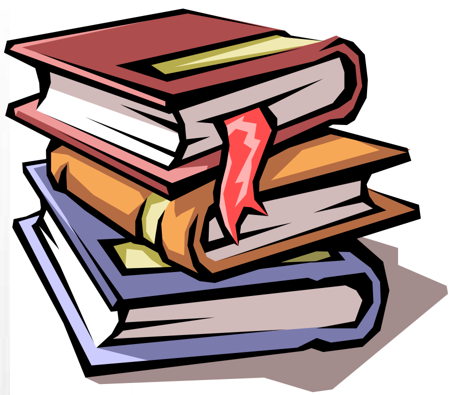
Topics

- 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

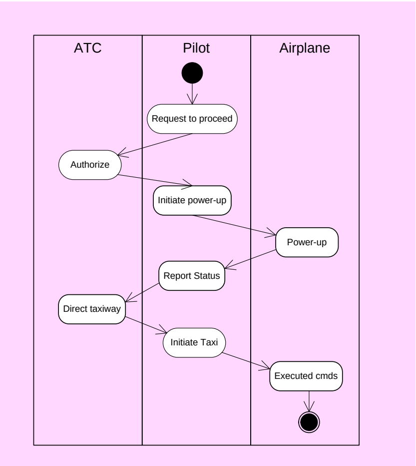
Systems Engineering: A Practice in Transition

Past



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

Future

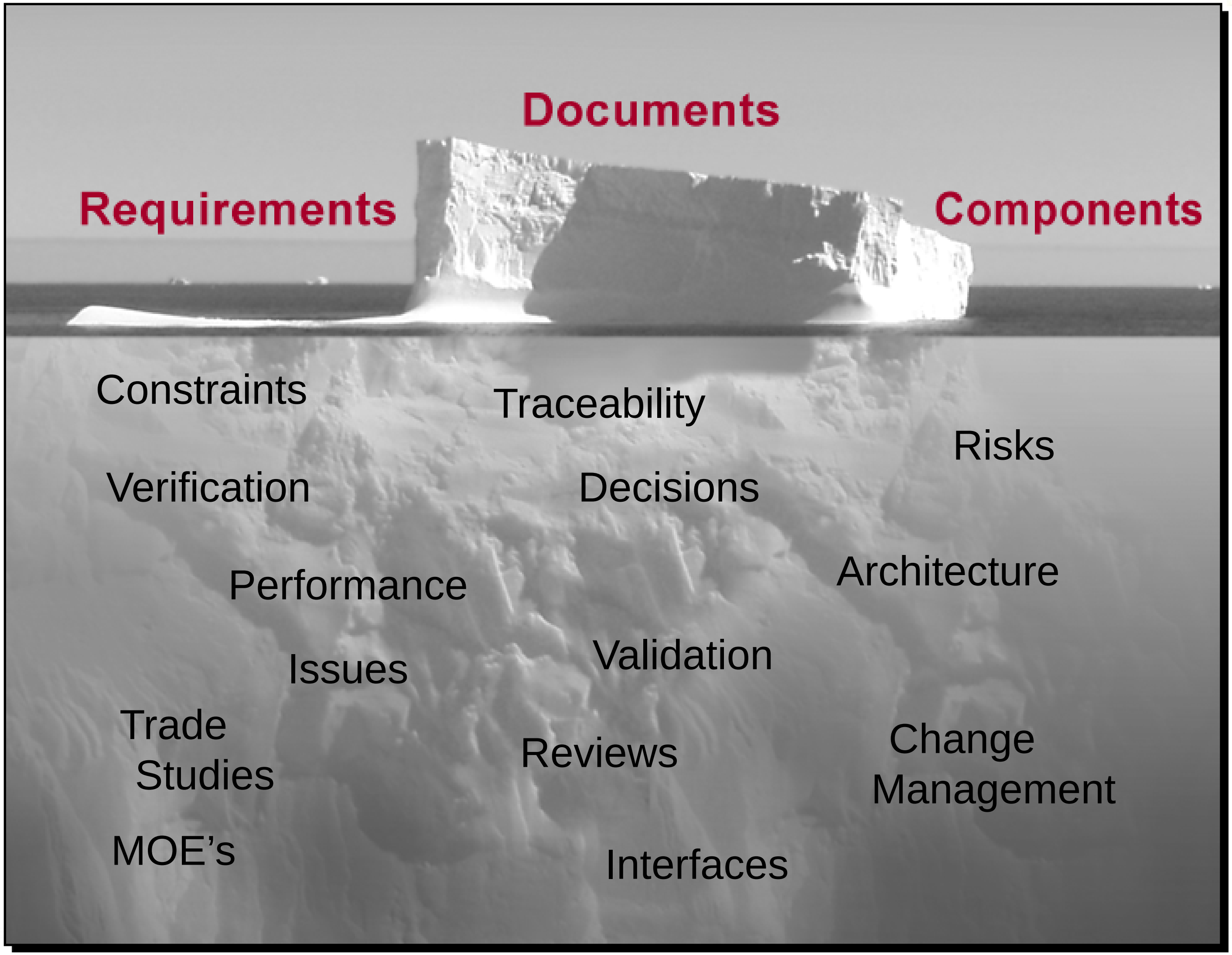


Moving from document-centric to model-centric

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

Systems Engineering: Broad Applicability

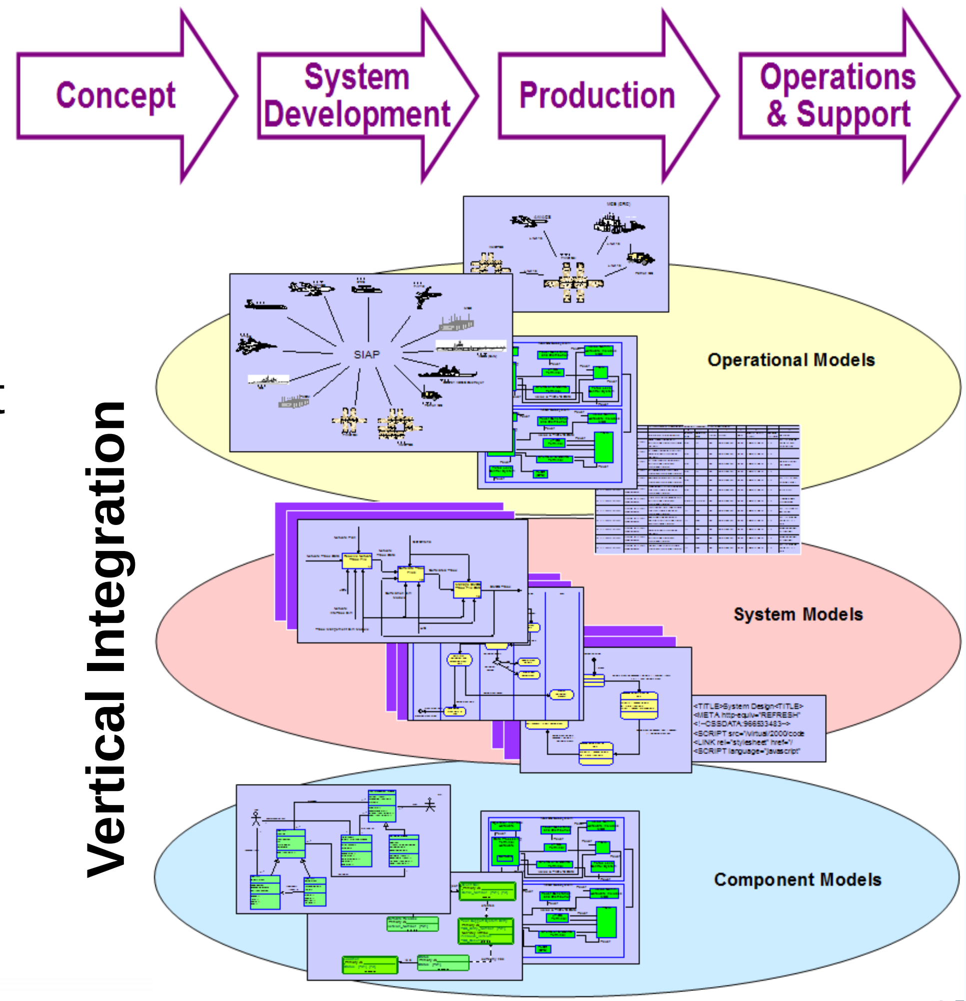
The Hidden Complexity of Systems Engineering



Model-Based Systems Engineering

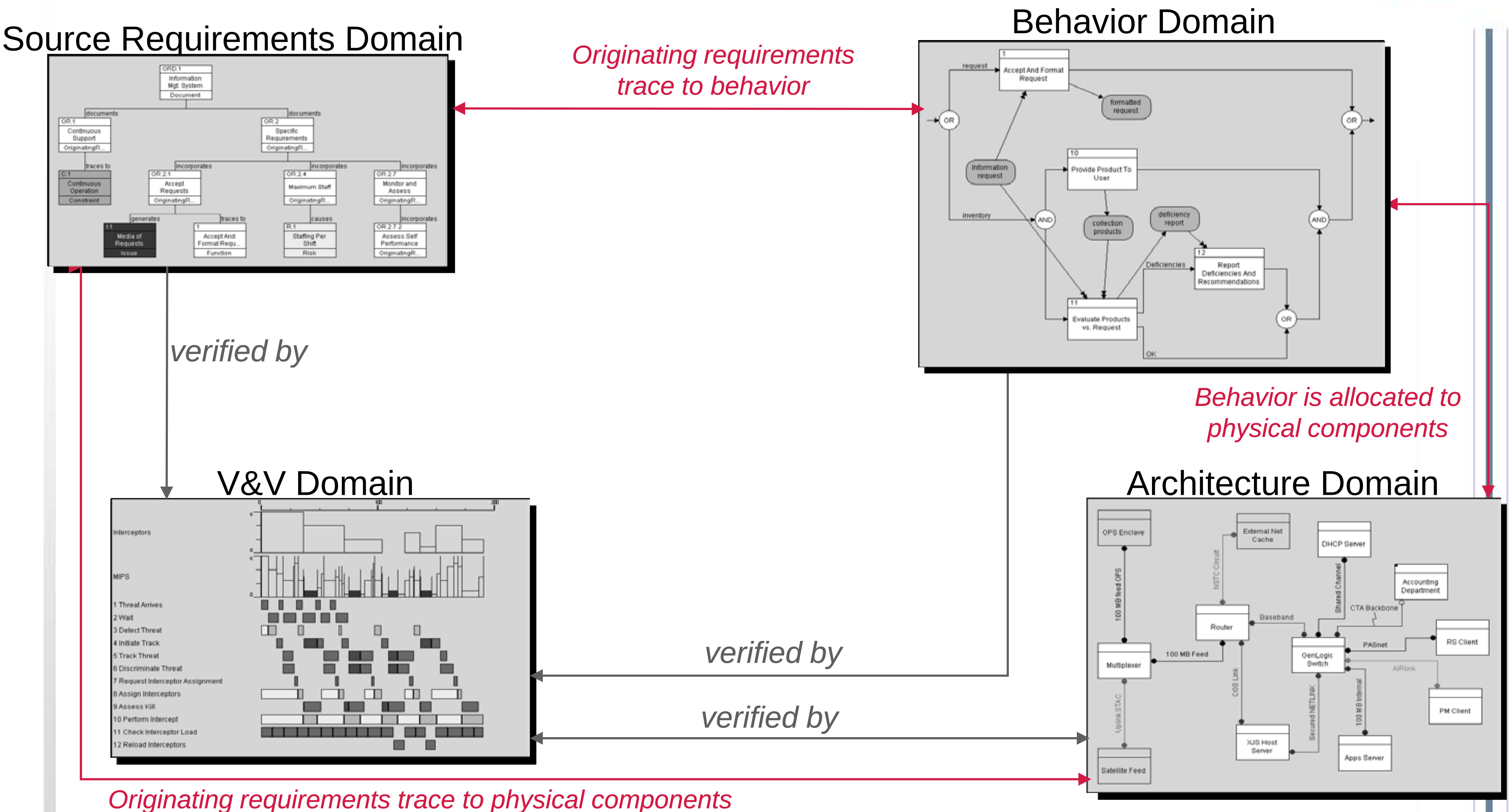
- Formalizes the practice of systems engineering through the use of models
- Broad in scope
  - Integrates with multiple modeling domains across life cycle from SoS to component
- Results in quality/productivity improvements & lower risk
  - Rigor and precision
  - Communications among system/project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

Life Cycle Support



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

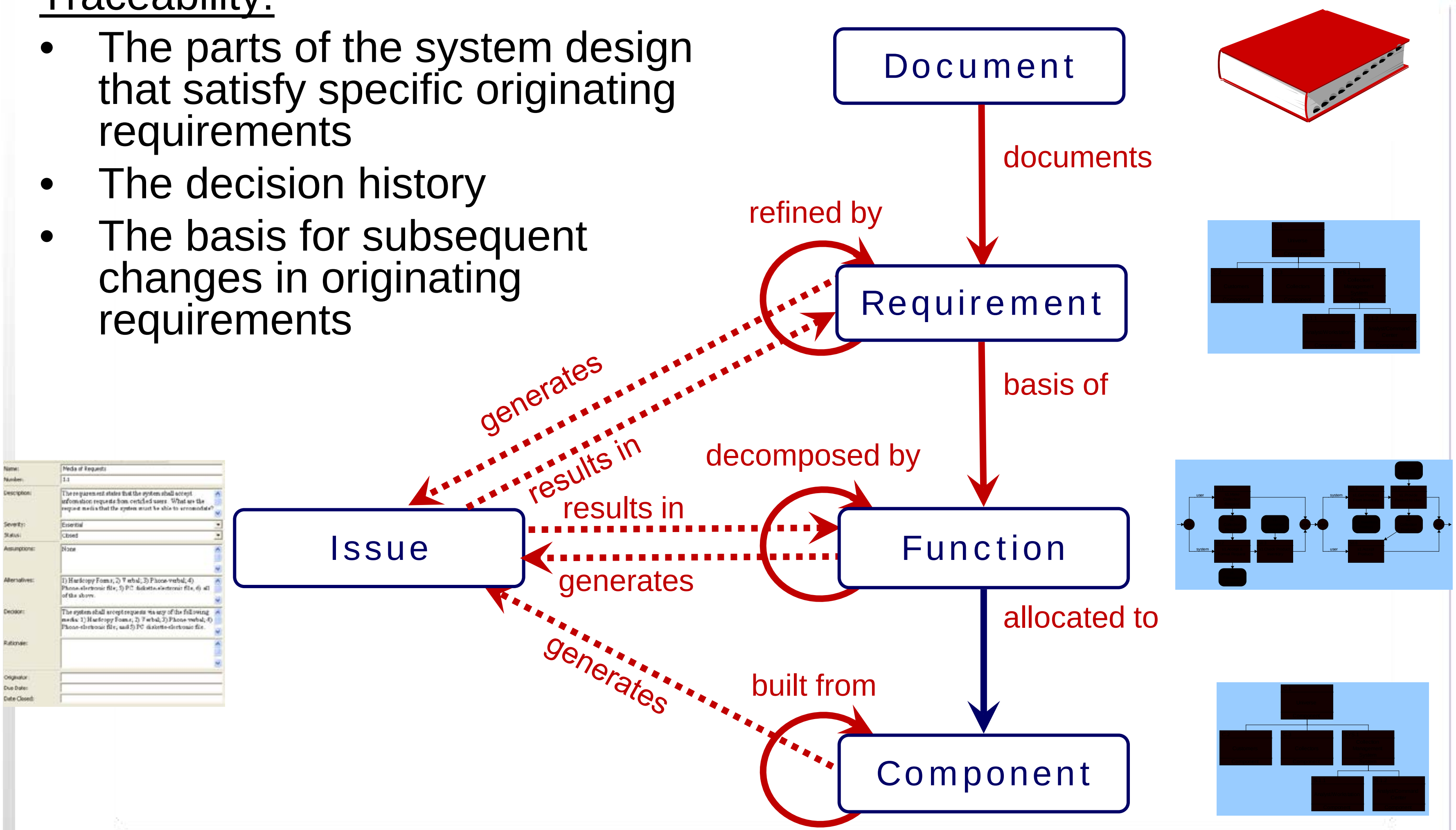
Setting the Context – The Four Primary Systems Engineering Activities



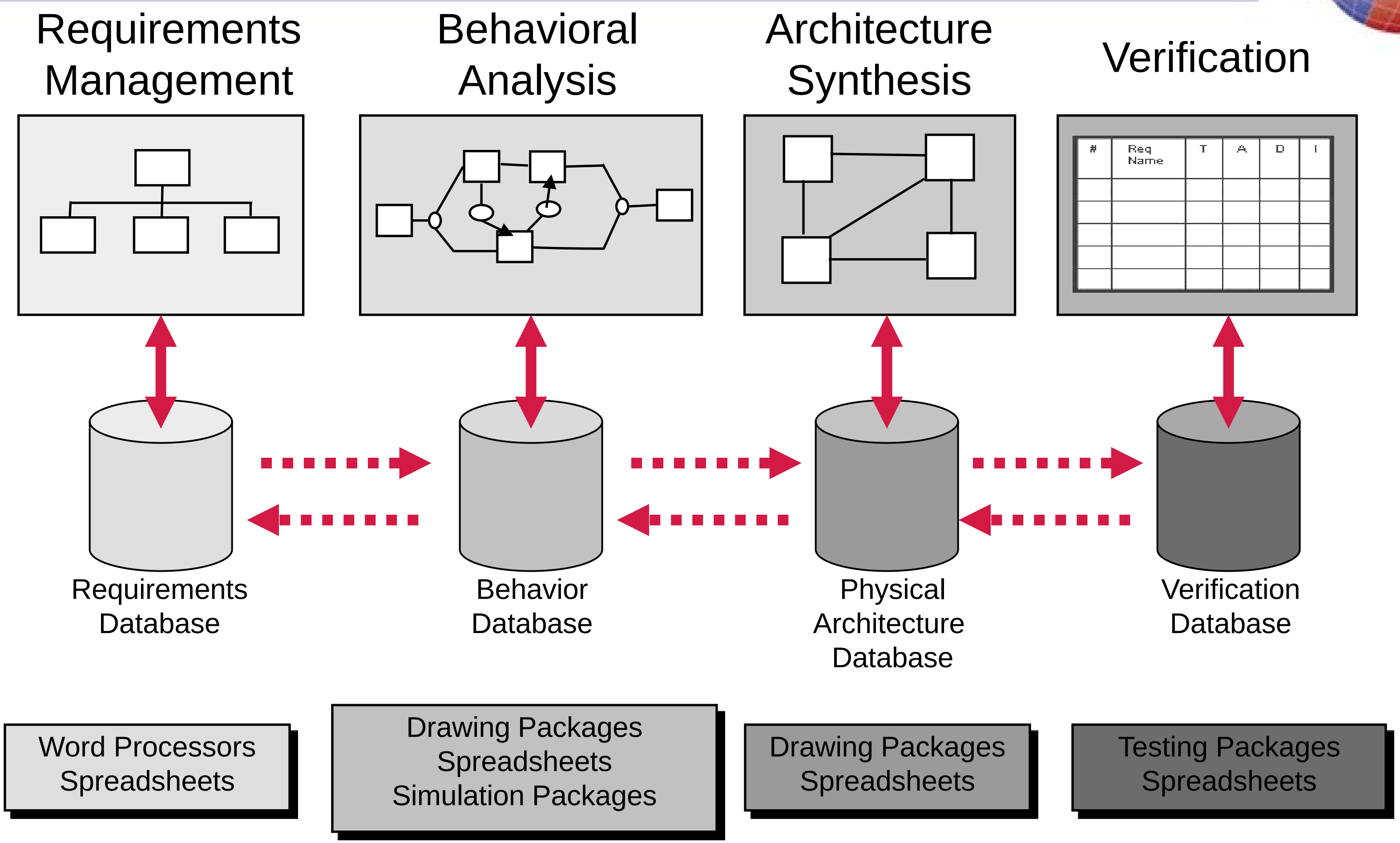
Primary Design Traceability; It's Done with Relationships (Verbs), Not Attributes (Adjectives)

<u>Traceability:</u>

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

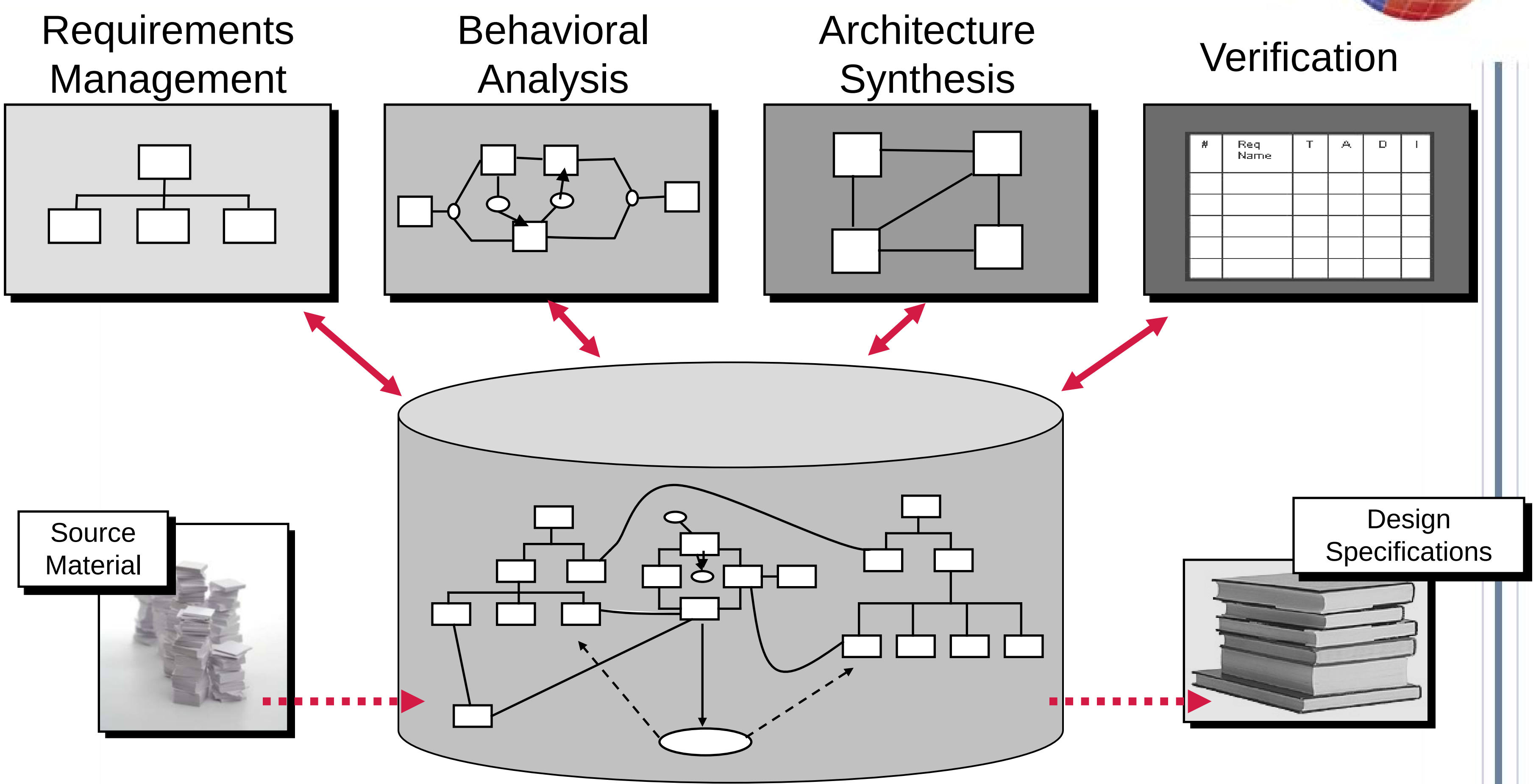


Common SE "Tool Suite" Architecture



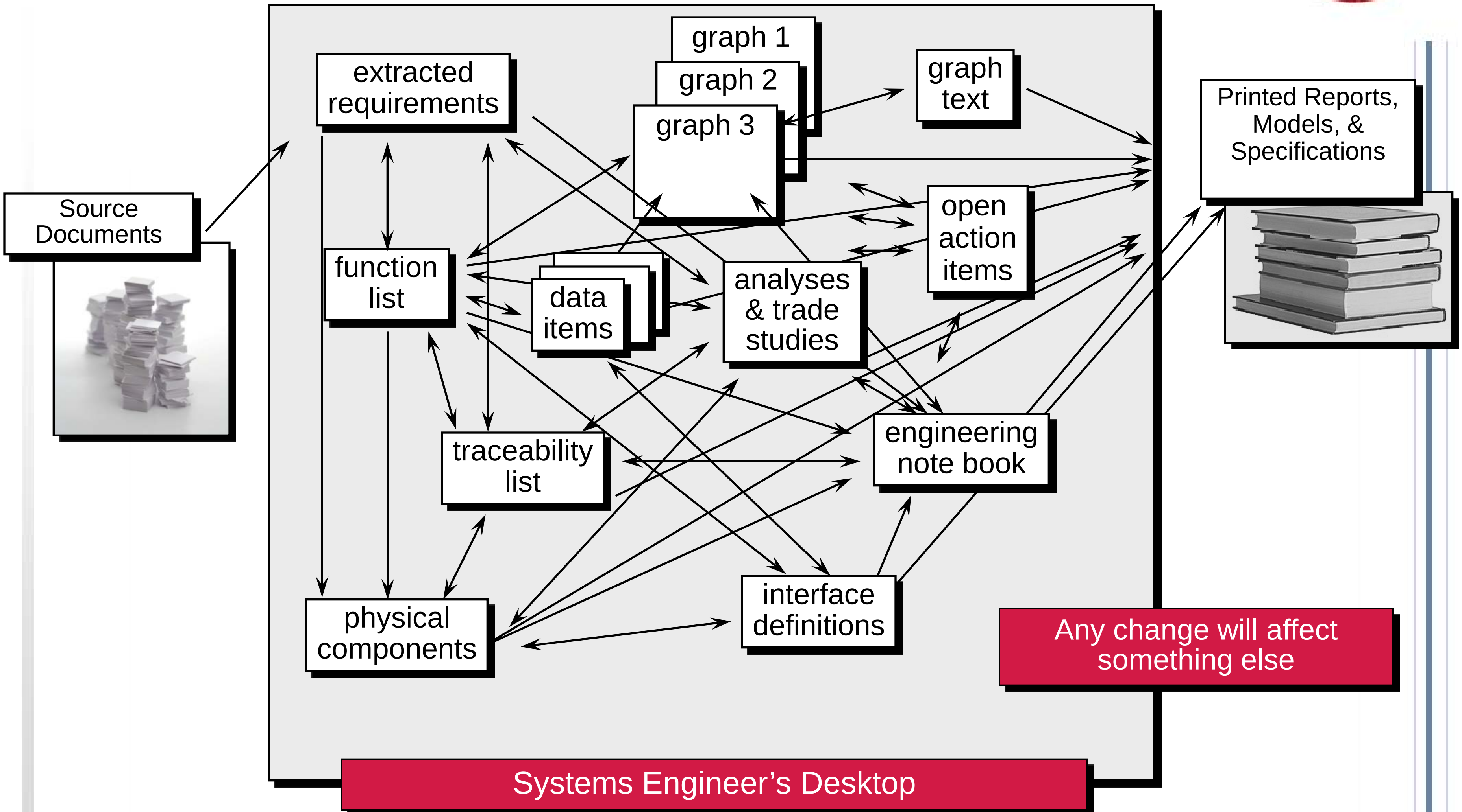
Multiple products utilizing independent databases forces extraordinary data management – and complicates the original SE effort

The Preferred SE Tool Architecture



Integrated, Consistent Analysis: Diagrams, complete specifications, and project work products automatically generated from the integrated model

The Systems Engineer's Dilemma: Integration and Synchronization



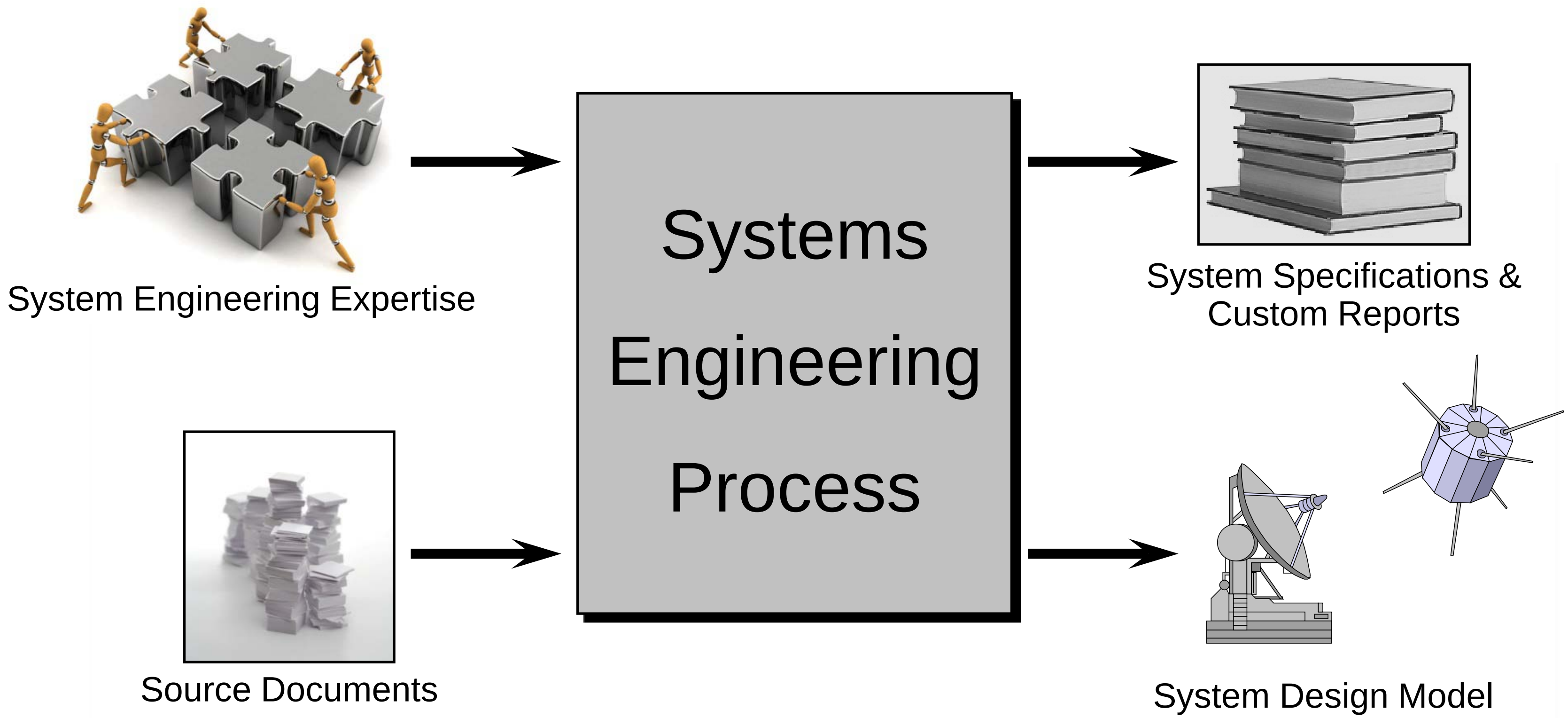
Why are there Problems with SE as Commonly Practiced Today?

- 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 Model-Based System Engineering Process: Its Inputs and Its Outputs



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

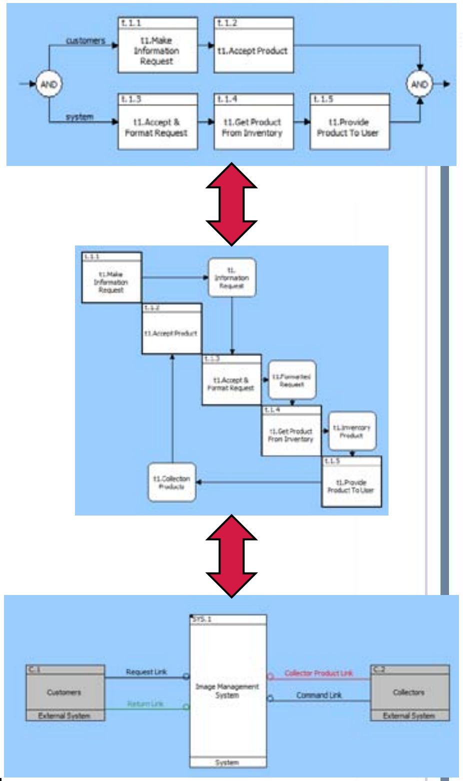
- Convergent
- Model-based
- Concurrent engineering
- Layered, hierarchical solution
- Central engineering repository
  - Incorporating system definition language
    - With graphical control constructs
    - Executable, dynamic validation of system logic
    - Context free
- Different initial conditions
  - 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)

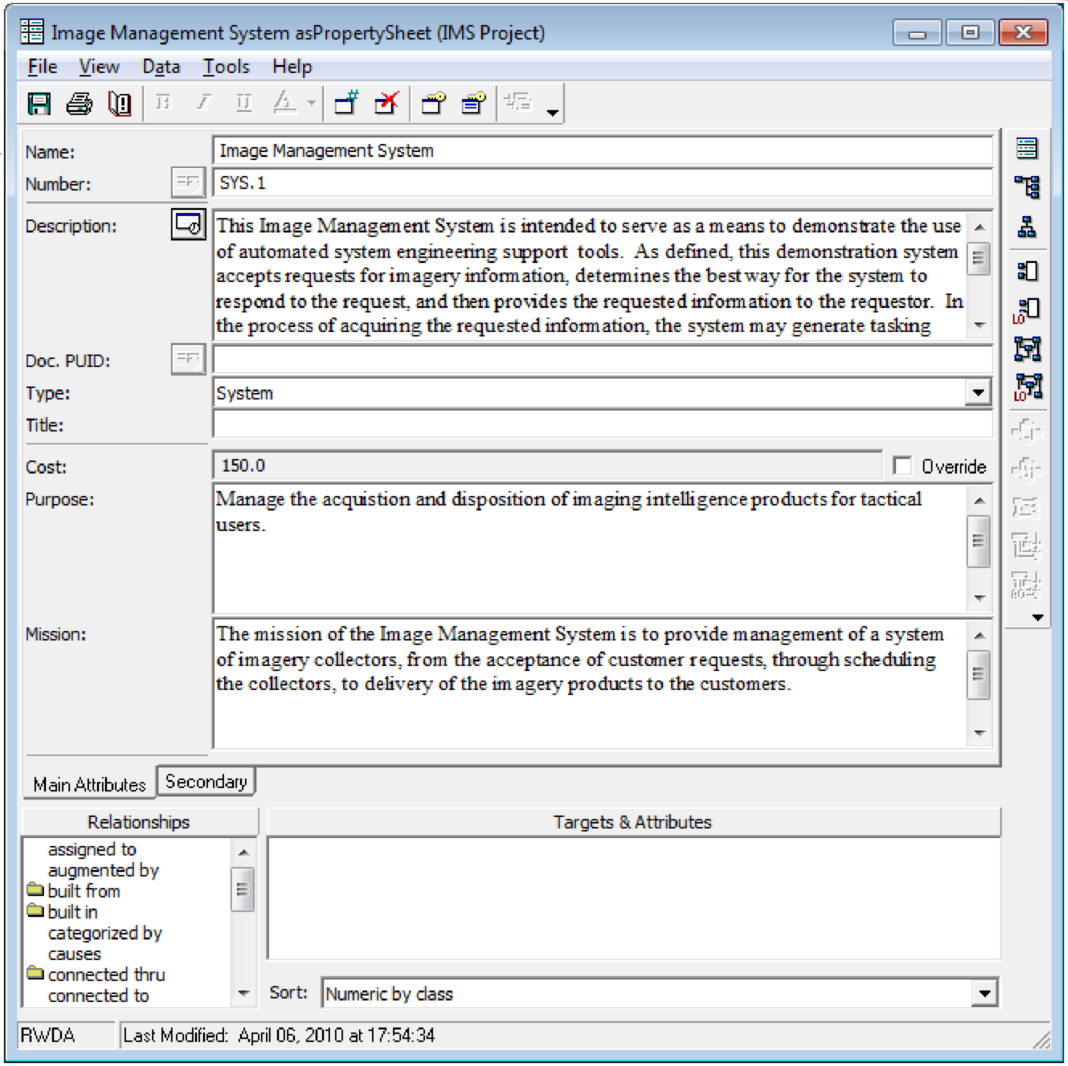
SDL Language	English Equivalent	MBSE Example
Element	**Noun**	• Requirement: **Place Orders** • Function: **Cook Burgers** • Component: **Cooks**
Relationship	**Verb**	• Requirement **<u>basis of</u>** Functions • Functions **are <u>allocated to</u>** Components
Attribute	**Adjective**	• **Creator** • **Creation Date** • **Description**
Attribute of Relationship	**Adverb**	Resource **<u>consumed by</u>** Function • **Amount (of Resource)** • **Acquire Available (Priority)**
Structure	**N/A**	• **Viewed as Enhanced FFBD or Activity Diagram**

Model Element Property Sheet (Representation in the Repository)

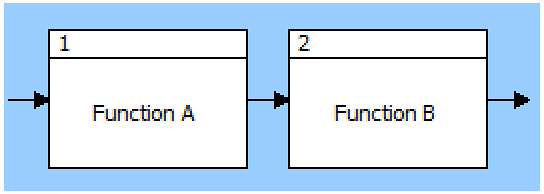
Element Name

Other Element Attributes

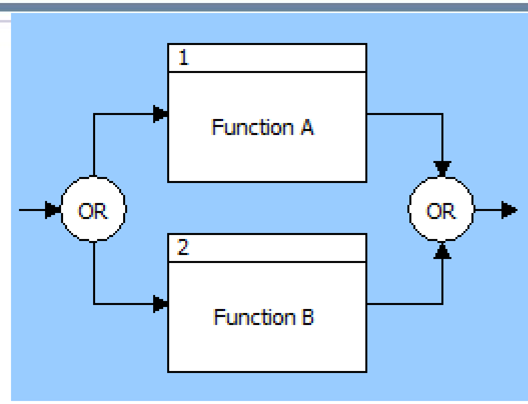
Element Relationships



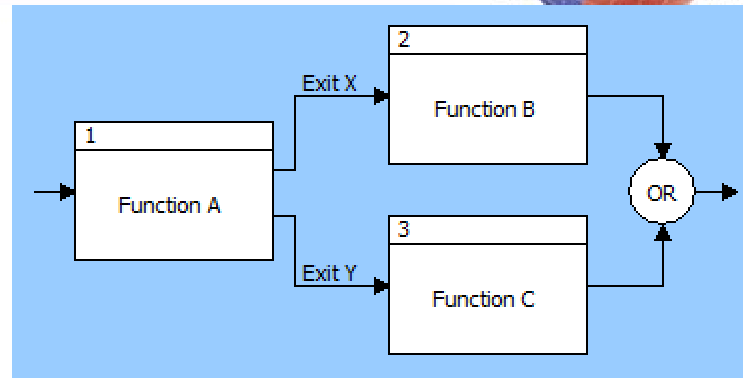
A Set of Complete and Executable Graphical Constructs (Structured Representations)



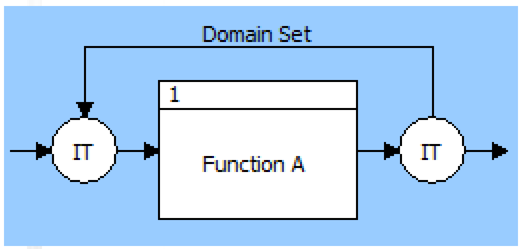
SEQUENCE



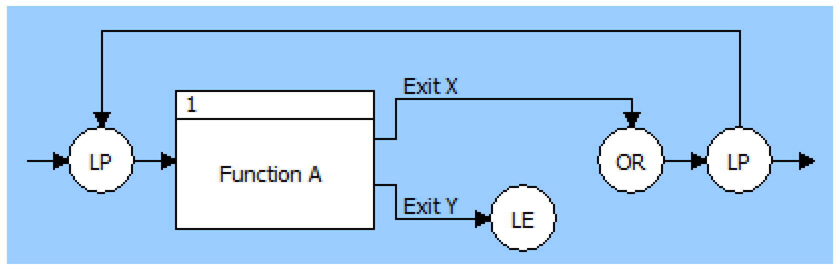
SELECT



MULTIPLE-EXIT

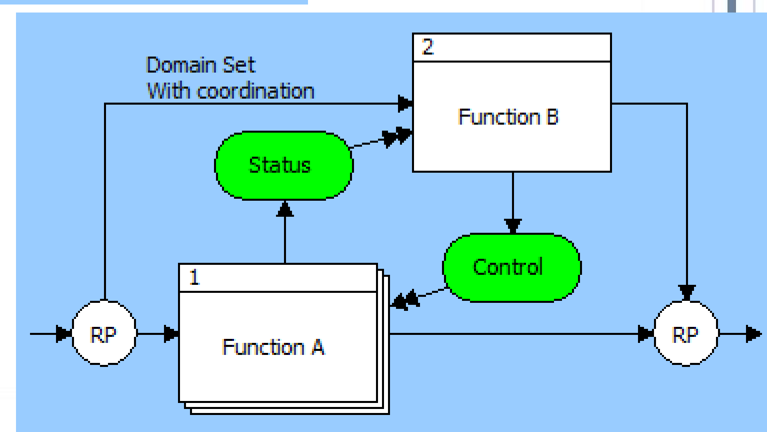


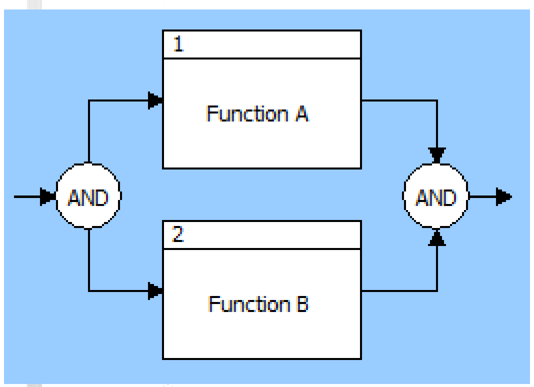
ITERATE



LOOP

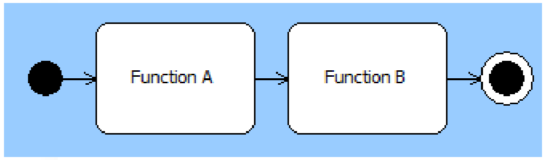
REPLICATE



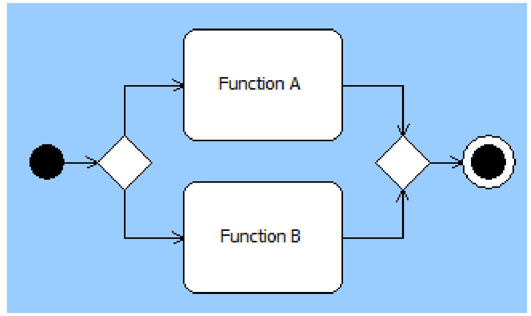


CONCURRENCY

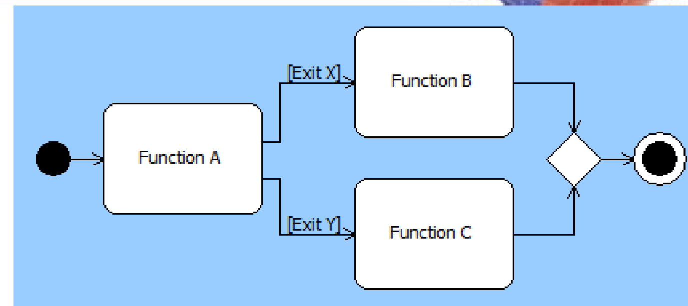
A Set of Complete and Executable Graphical Constructs (SysML Representations)



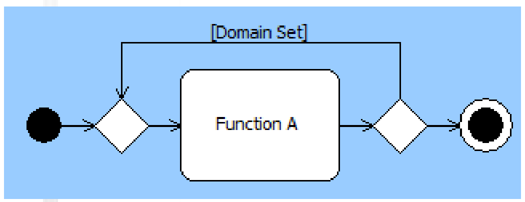
SEQUENCE



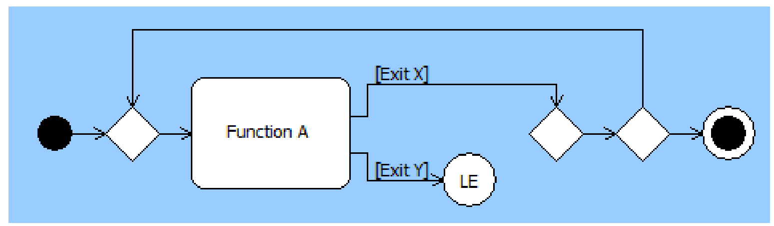
SELECT



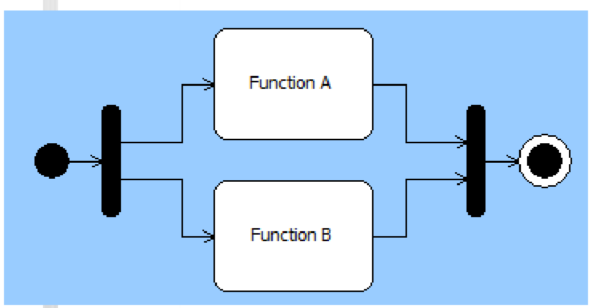
MULTIPLE-EXIT



ITERATE

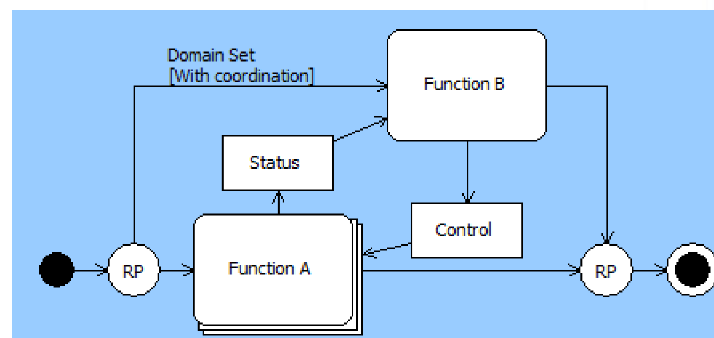


LOOP



CONCURRENCY

REPLICATE

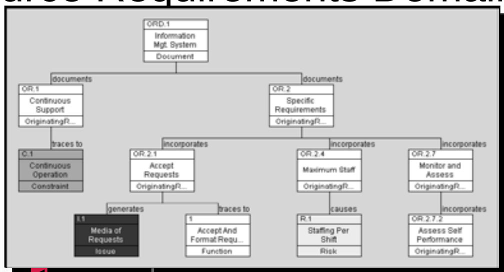


The Power of Models and Graphical Representations

- 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

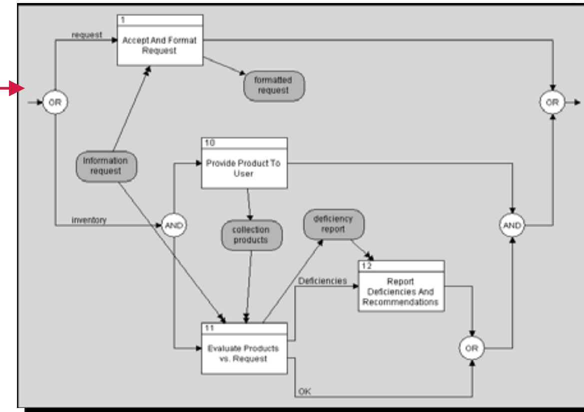
Integrating the Four Primary SE Activities through a Design Repository

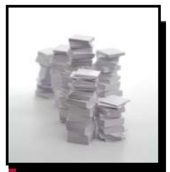
Source Requirements Domain



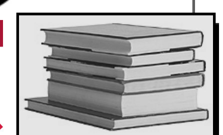
Originating requirements trace to behavior

Behavior Domain



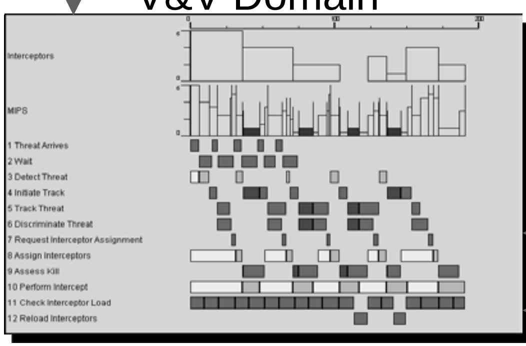


System Design Repository



verified by

V&V Domain

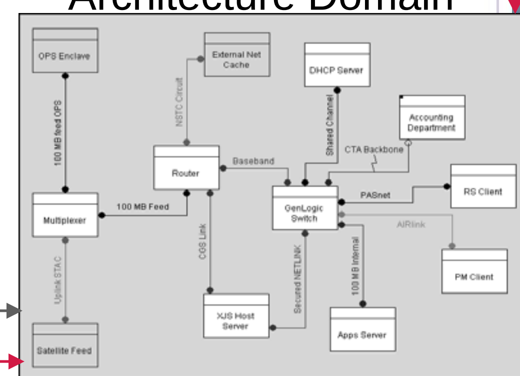


verified by

verified by

Behavior is allocated to physical components

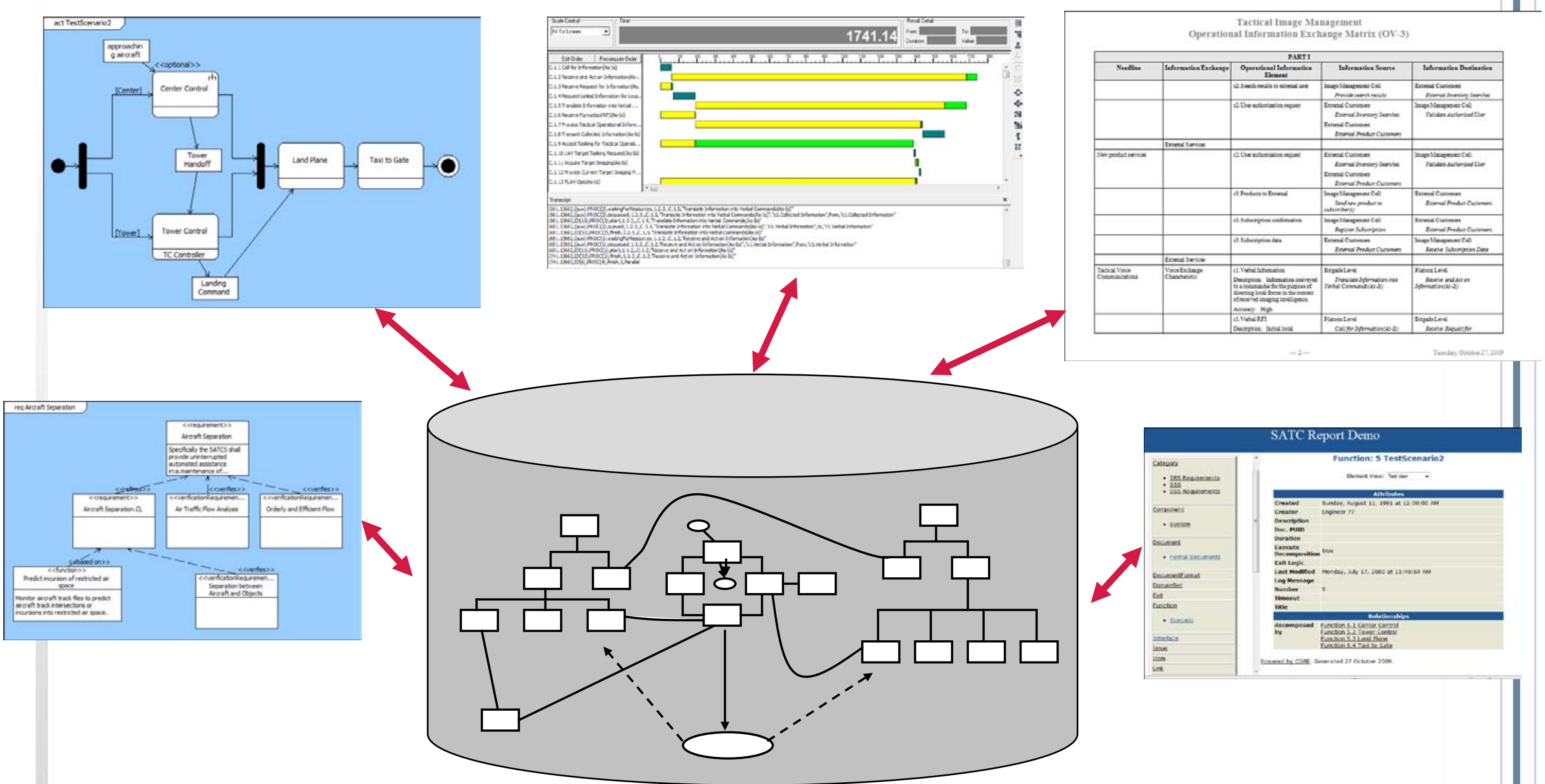
Architecture Domain



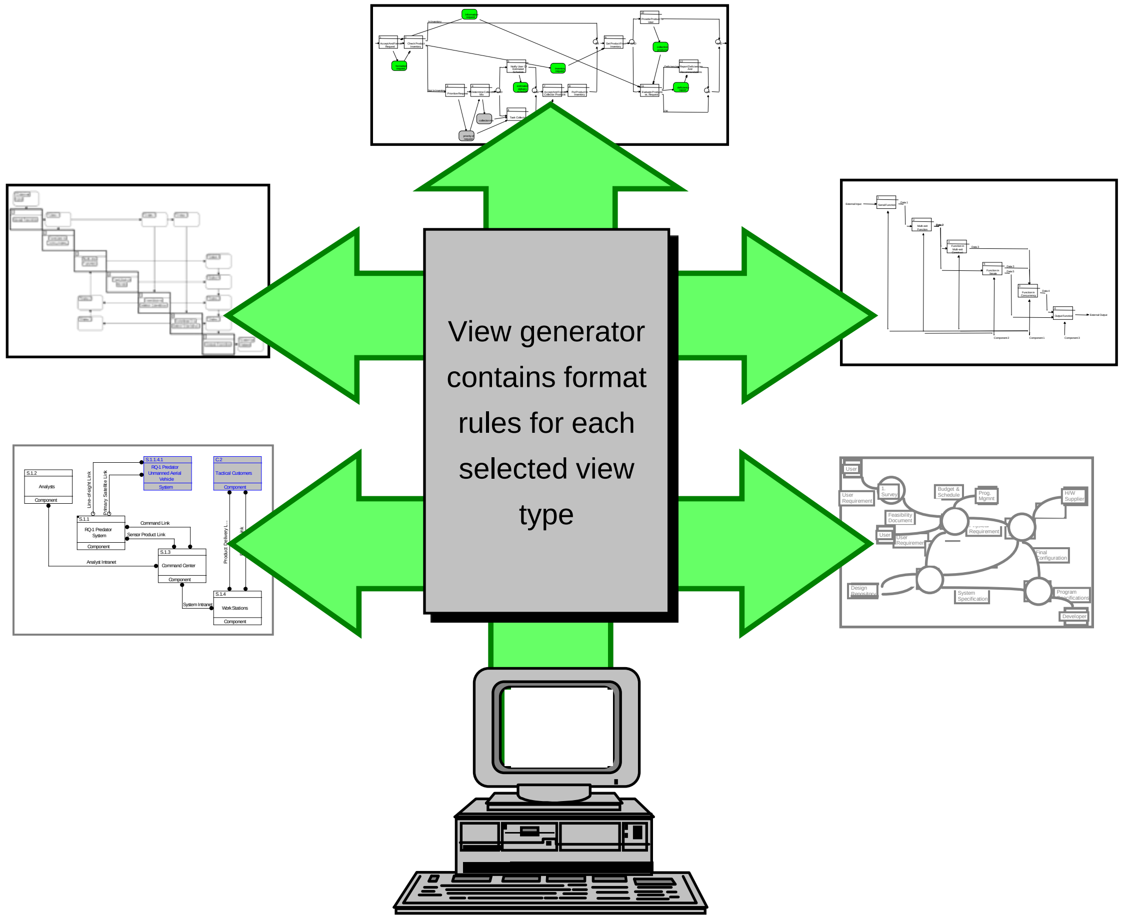
Originating requirements trace to physical components

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



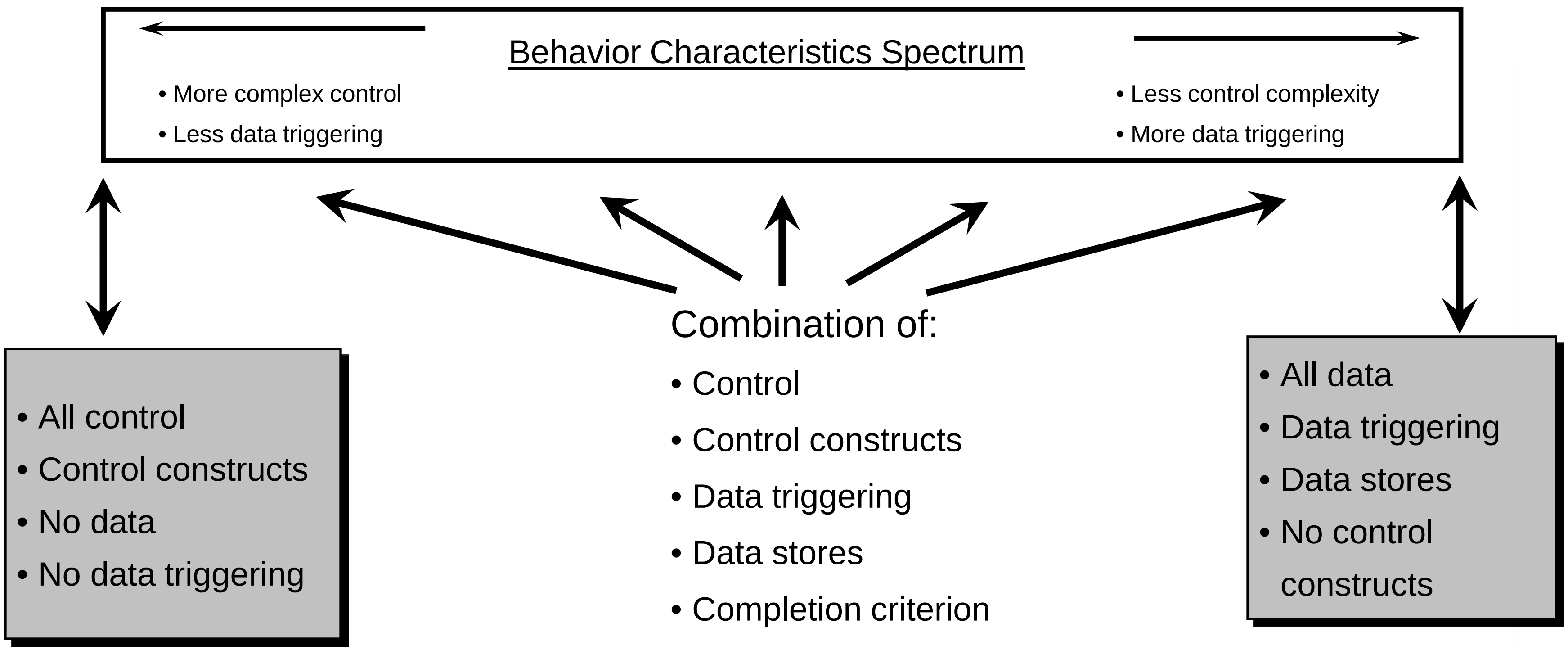
View Generators using a Common Repository Guarantee Consistent Views



- 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

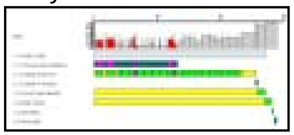
Views are projections of the model. Choose the views that serve the purpose.

A Momentary Aside for Some Insight – The Control Enablement & Data Triggering Spectrum



Relationships of the Graphical Representations - FFBDs & DFDs are Limiting Cases

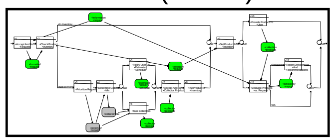
Dynamic Timelines



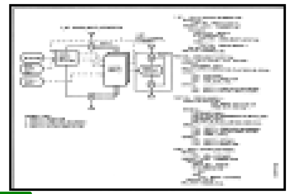
EFFBD / Behavior Diagrams

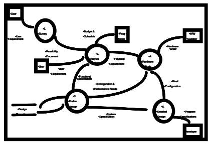
- Provide Both Data Triggering and Control Constructs
- Balance Depends on Needs and Analyst
- Diagram is Executable

EFFBD (CORE)



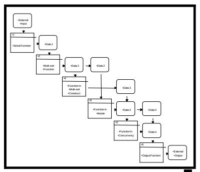
BD (DCDS)





Data Flow Diagram

- Only Data Triggering
- No Control Constructs



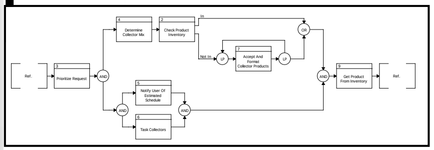
N2 Chart

- Equivalent to DFD

Behavior Characteristics Spectrum

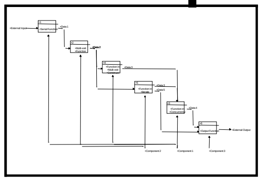
- More complex control
- Less data triggering

- Less control complexity
- More data triggering



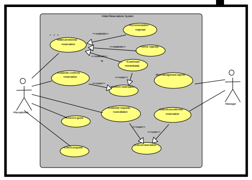
Function Flow Block Diagram

- Only Control Constructs
- No Data Triggering



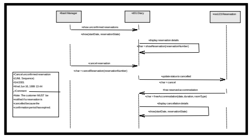
IDEF0 Diagram

- Primarily DFD
- Some Control, No Control Constructs



Use Case

- Equivalent to DFD



Sequence Diagram

- Message Flows

Applying an MBSE Process

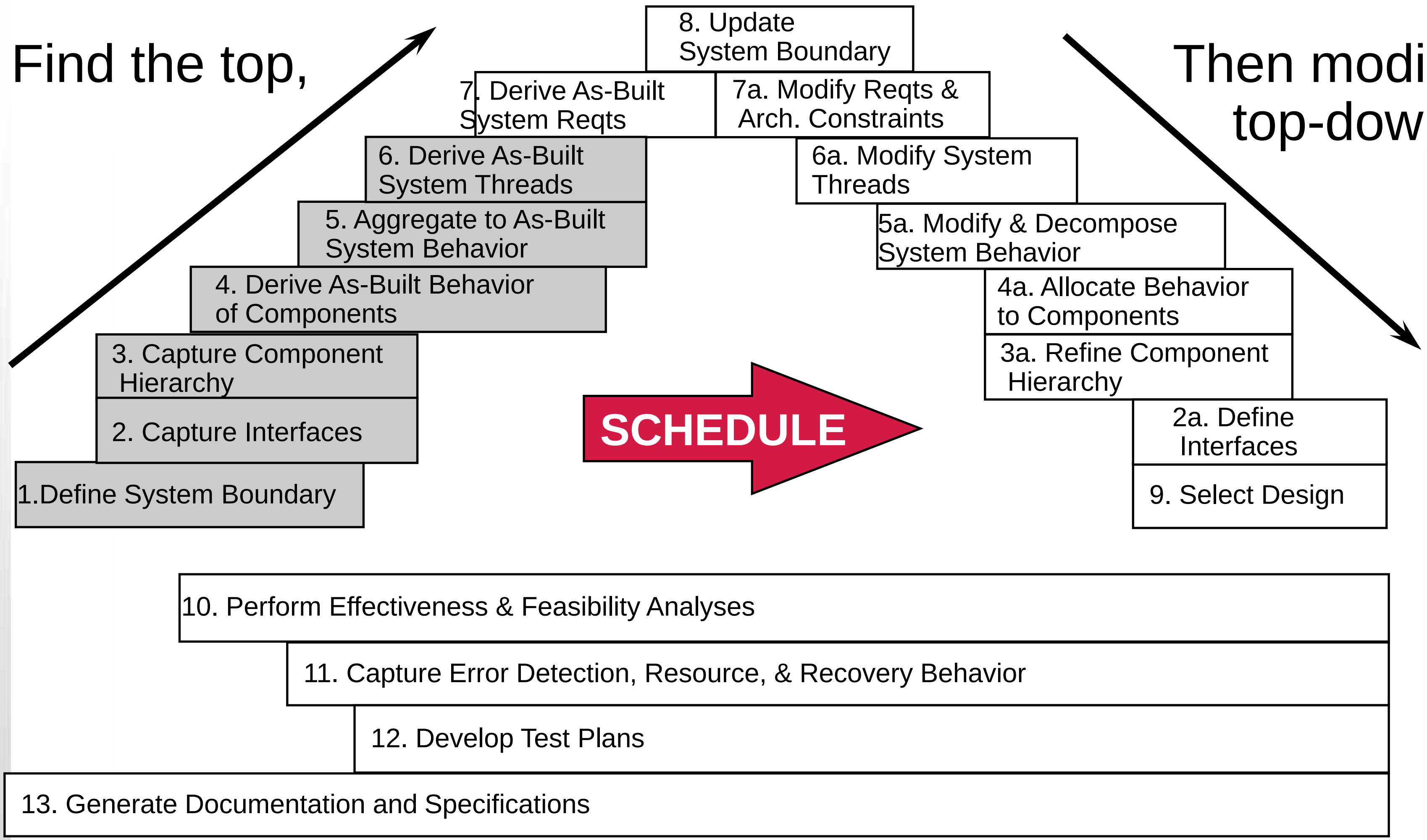
Model-Based Systems Engineering Activities Timeline – Top Down

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

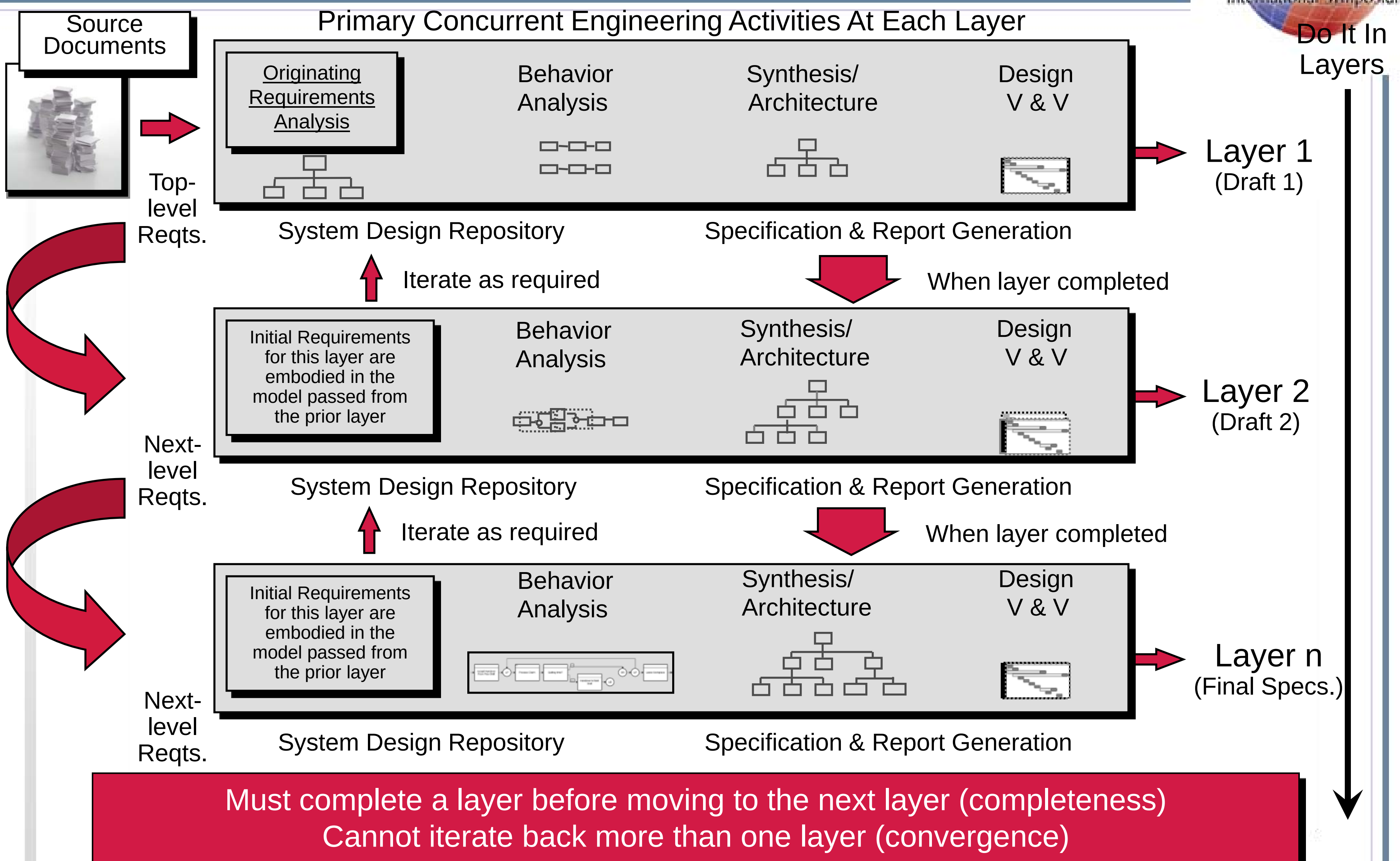
Activity bars represent movement of "center of gravity" of systems engineering team.
Concurrent engineering is assumed.

SCHEDULE

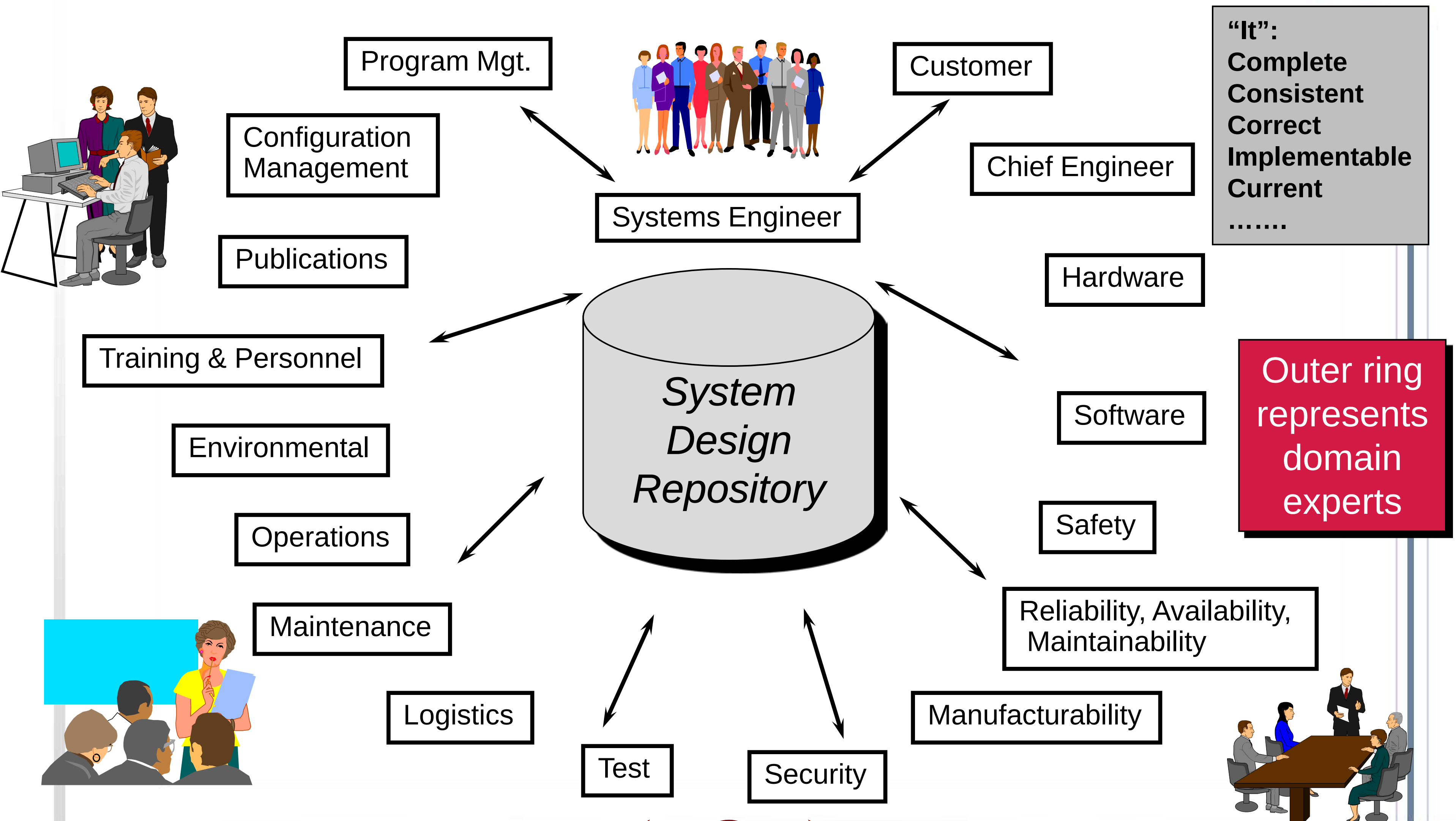
Model-Based System Engineering Activities Timeline – Reverse Engineering



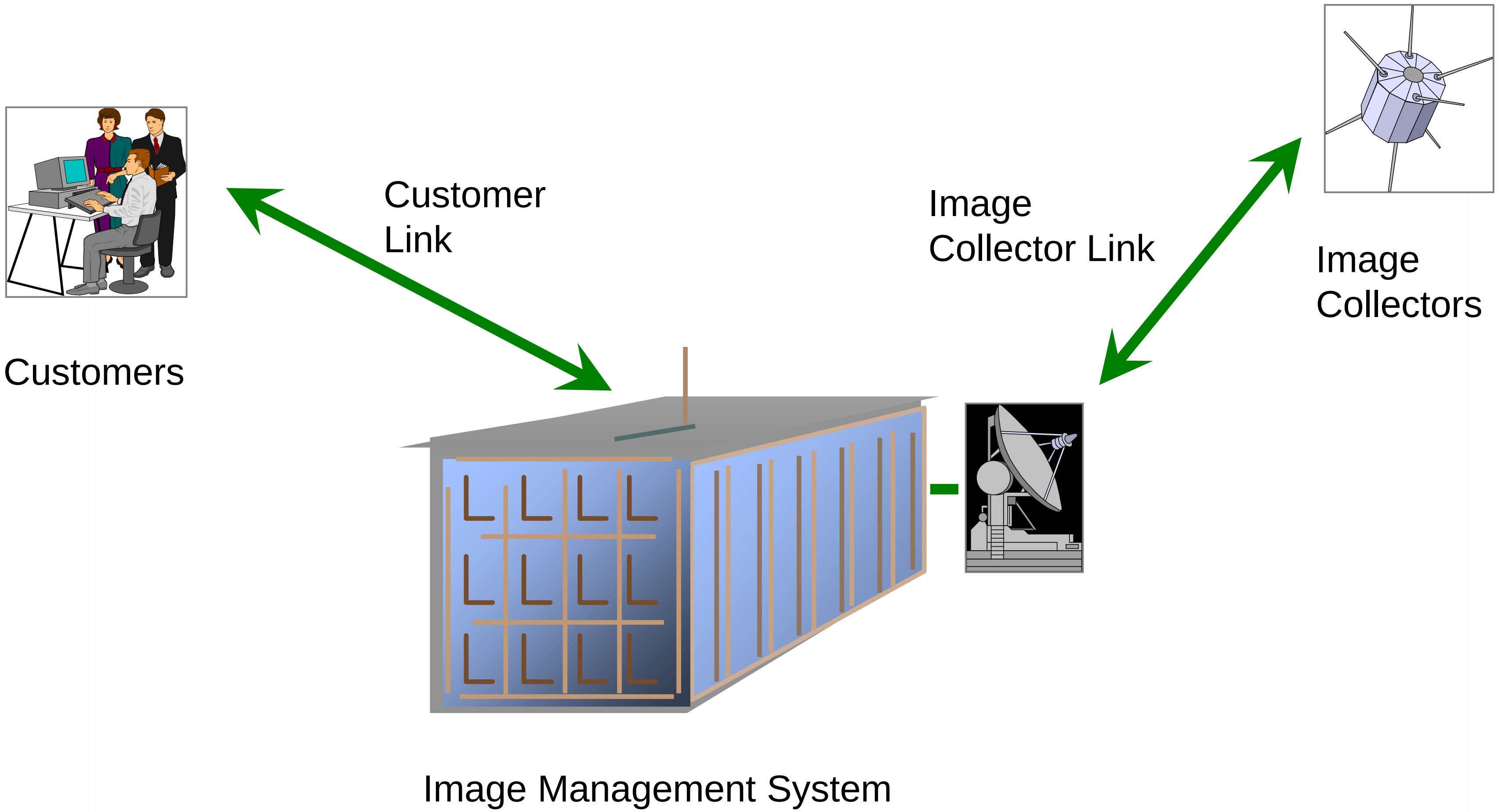
MBSE – the Onion Model
Doing Systems Engineering in Increments/Layers



Concurrent Engineering Enables “Design It In” Don’t Try to Test It In, Review It In, Annotate It In …



The Image Management System (IMS) Overview



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

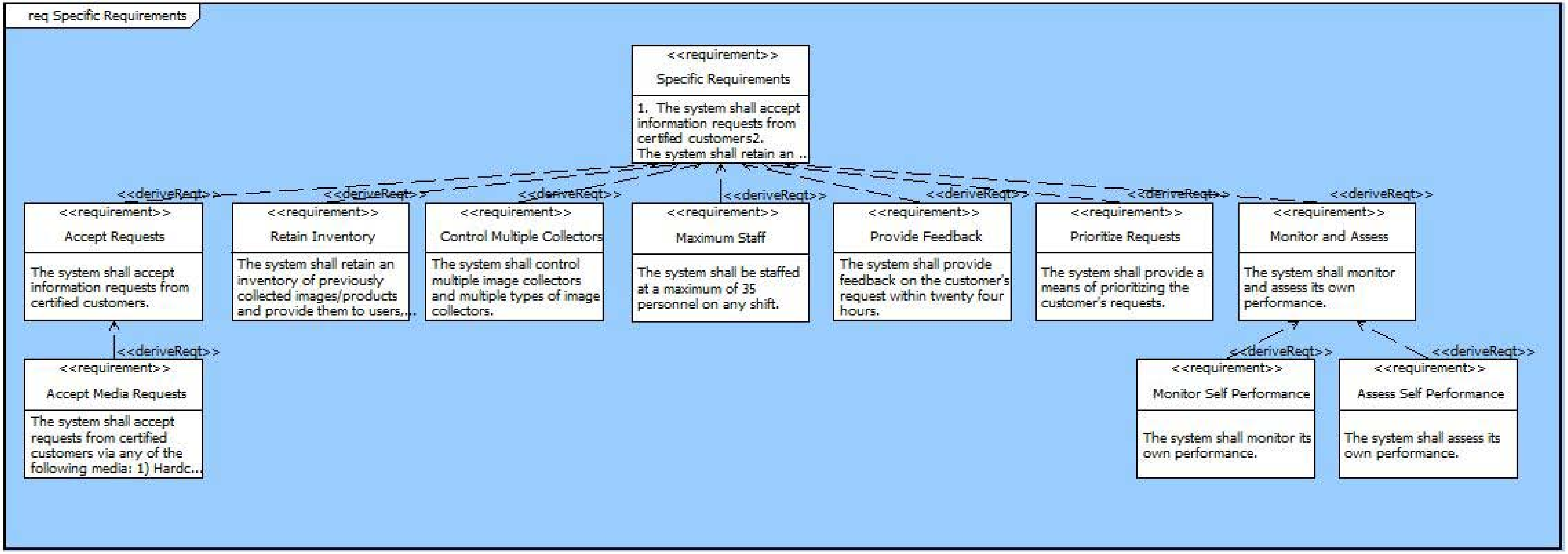
“Making Sure That We Solve the Right Problem”

- 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

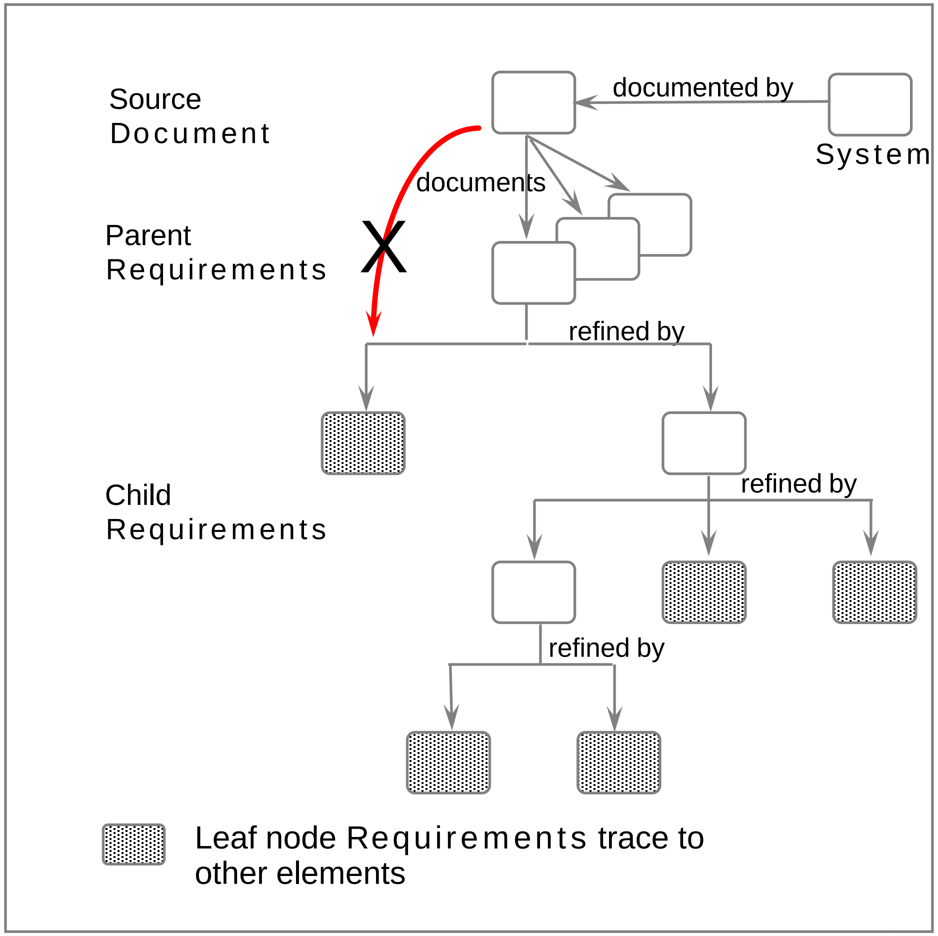
- System Concept Paper
- Executive Order
- Concept of Operations
- Statement of Work
- Vendor Package/Contract
- Preliminary Specification
- Change Request
- Trade Study Report
- Standards (MIL-STD or Commercial)
- Meeting Minutes'
- Business Plan
- Market Analysis

Capturing the System Requirements



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 <u>documents/documented by</u> relationships.
- Obtain traceability between parent/child Requirements with the <u>refines/refined by</u> relationships.



Issues

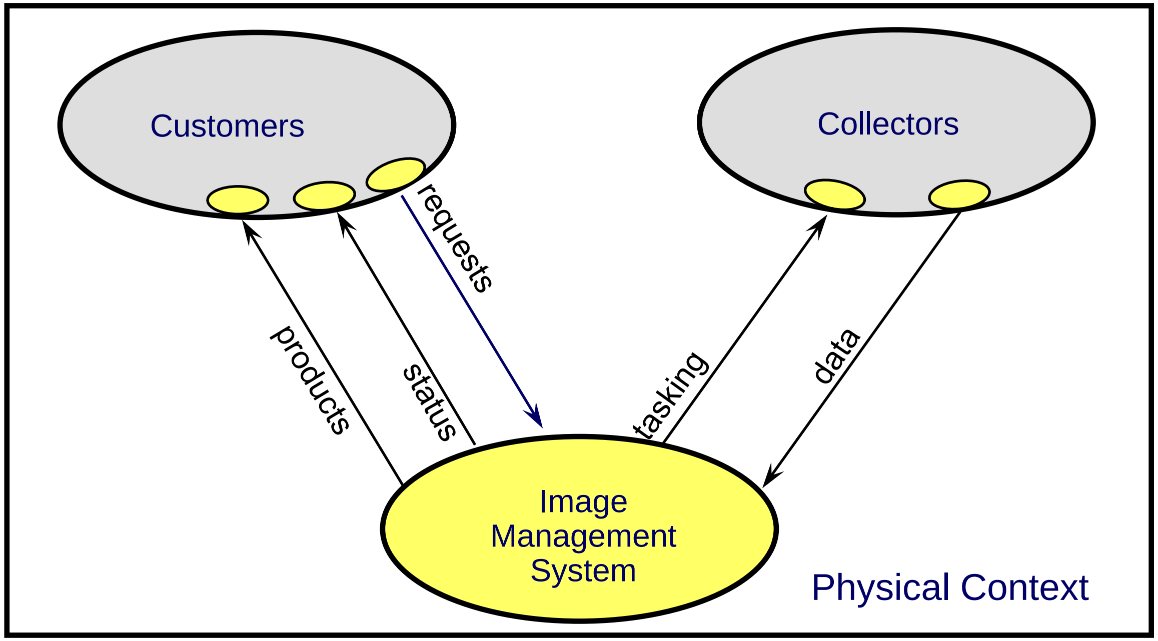
- During the requirements capture and analysis process, it is likely that problems will be found
  - Requirements are not clear or complete
  - Requirements may contradict each other
  - Requirements may be over or under specified
- It is highly desirable to have a mechanism to capture these issues, 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



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

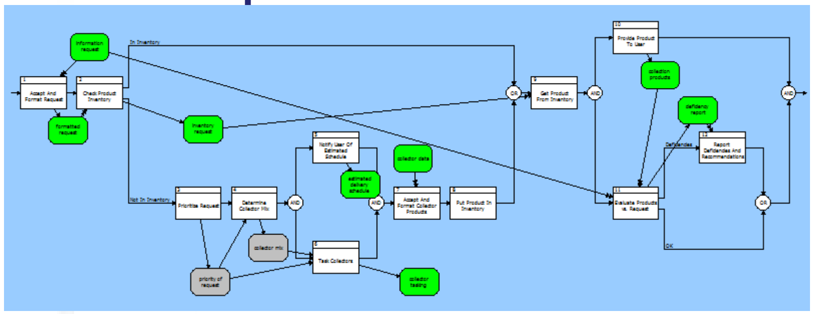
System Behavior

- 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

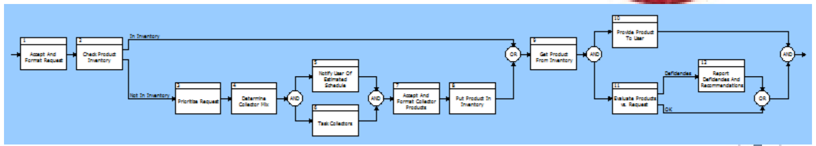
Behavior is essential for providing the complete systems engineering model of any system or process.

The Many Faces of Behavior

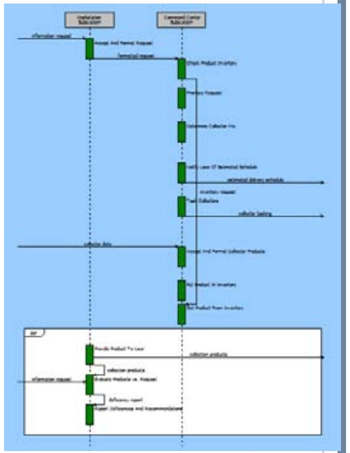
EFFBD or Activity Diagram
Complete and Executable



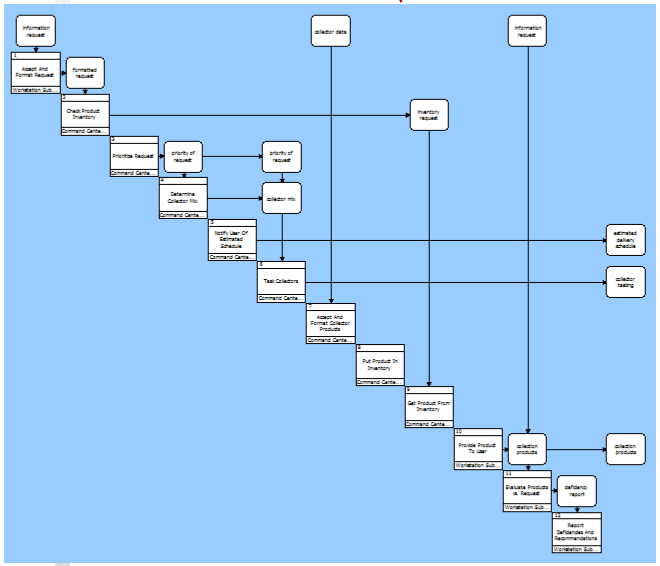
FFBD **Lacks data**



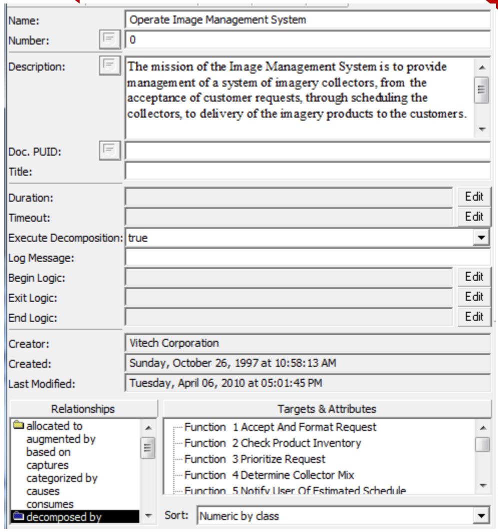
Sequence Diagram
Lacks structure



N2 Diagram
Lacks constructs



Property Sheet

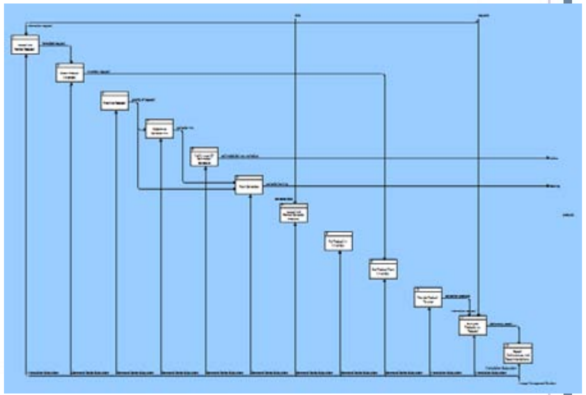


Name: Operate Image Management System
Number: 0
Description: The mission of the Image Management System is to provide management of a system of imagery collectors, from the acceptance of customer requests, through scheduling the collectors, to delivery of the imagery products to the customers.
Doc. PUID:
Title:
Duration: Edit
Timeout: Edit
Execute Decomposition: true
Log Message:
Begin Logic: Edit
Exit Logic: Edit
End Logic: Edit
Creator: Vitech Corporation
Created: Sunday, October 26, 1997 at 10:58:13 AM
Last Modified: Tuesday, April 06, 2010 at 05:01:45 PM

Relationships	Targets & Attributes
allocated to	Function 1 Accept And Format Request
augmented by	Function 2 Check Product Inventory
based on	Function 3 Prioritize Request
captures	Function 4 Determine Collector Mix
categorized by	Function 5 Notify User Of Estimated Schedule
causes	
consumes	
decomposed by	

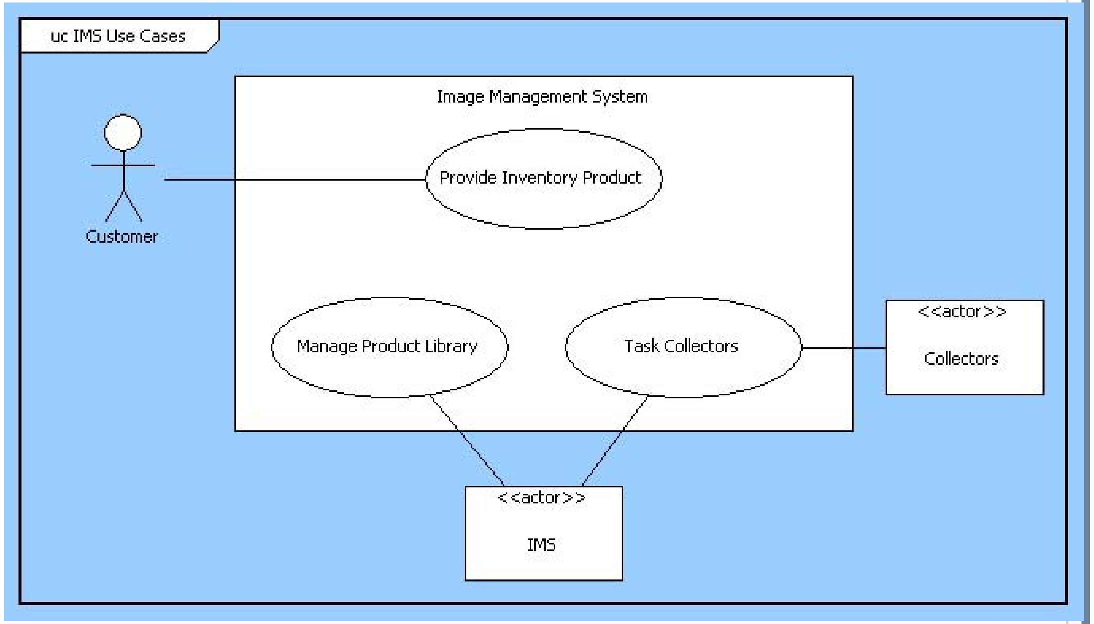
Sort: Numeric by class

IDEF0 **Lacks constructs**



Identifying Use Cases

- Derived from system context, operational concept, and requirements
- Should include
  - Preconditions
  - Primary flow
  - Alternate flow(s)
  - Post-conditions



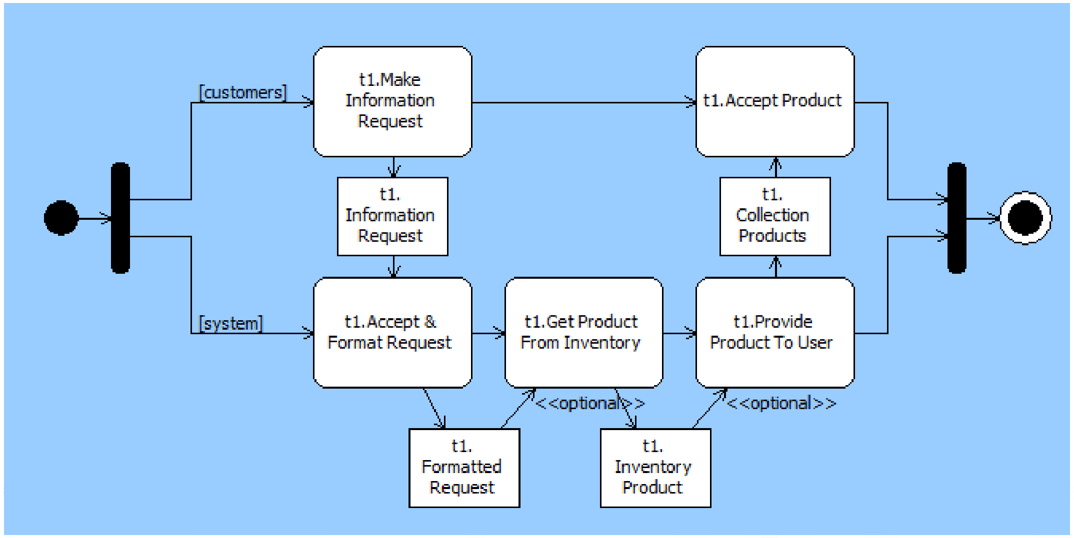
- Elaborated by system threads or system behavior

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

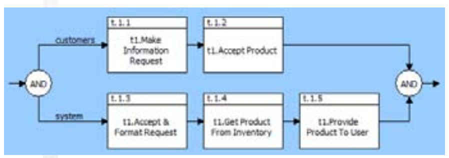
- Definition: A thread is a single stimulus/response behavior path through a system
- Threads give us insight on what the system must do (functions) to respond to different classes of system stimuli to produce desired outputs and behavior
- Thread derivation is a discovery process
- Thread derivation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



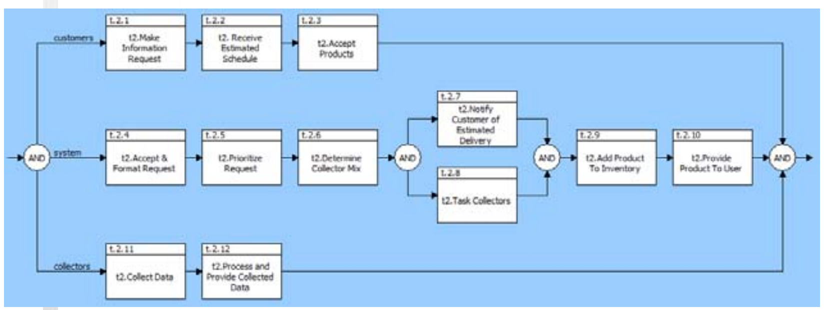
Activity Diagram

From Threads to Integrated Behavior or Operational Architecture

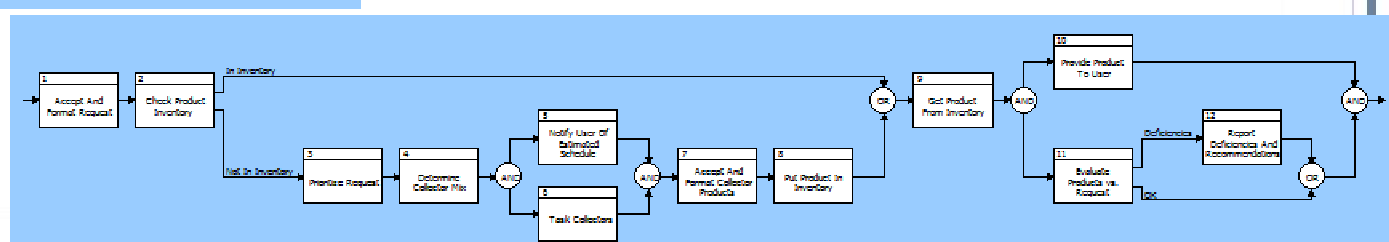
- Define distinct classes of threads based on system I/O
- Start with one simple thread per class of system input
- Preserve each thread (for thread testing, concept of operations, etc.)



1. Derive Threads



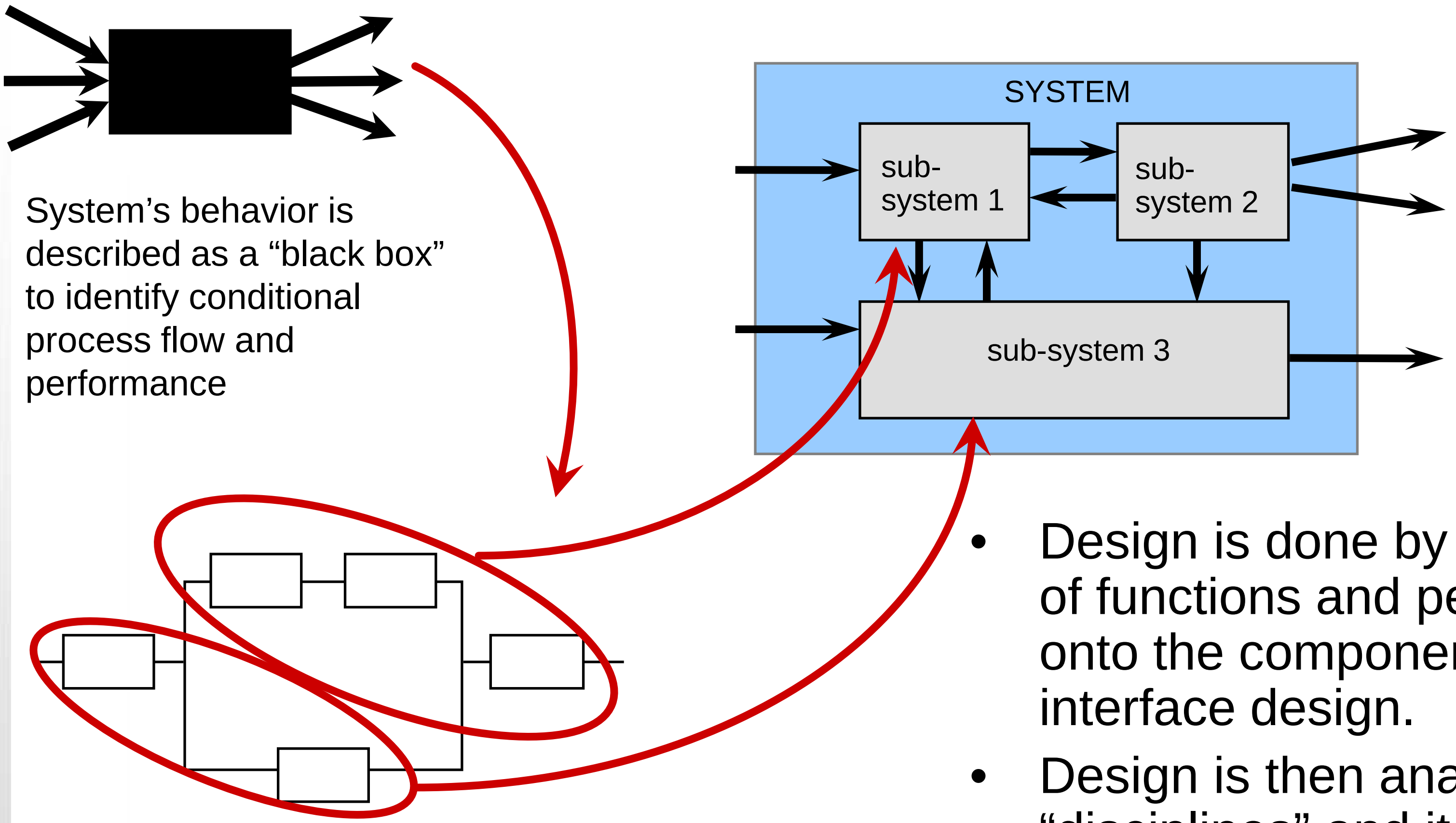
2. Integrate Threads to Define Integrated System Behavior



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

Key Concepts of Systems Engineering

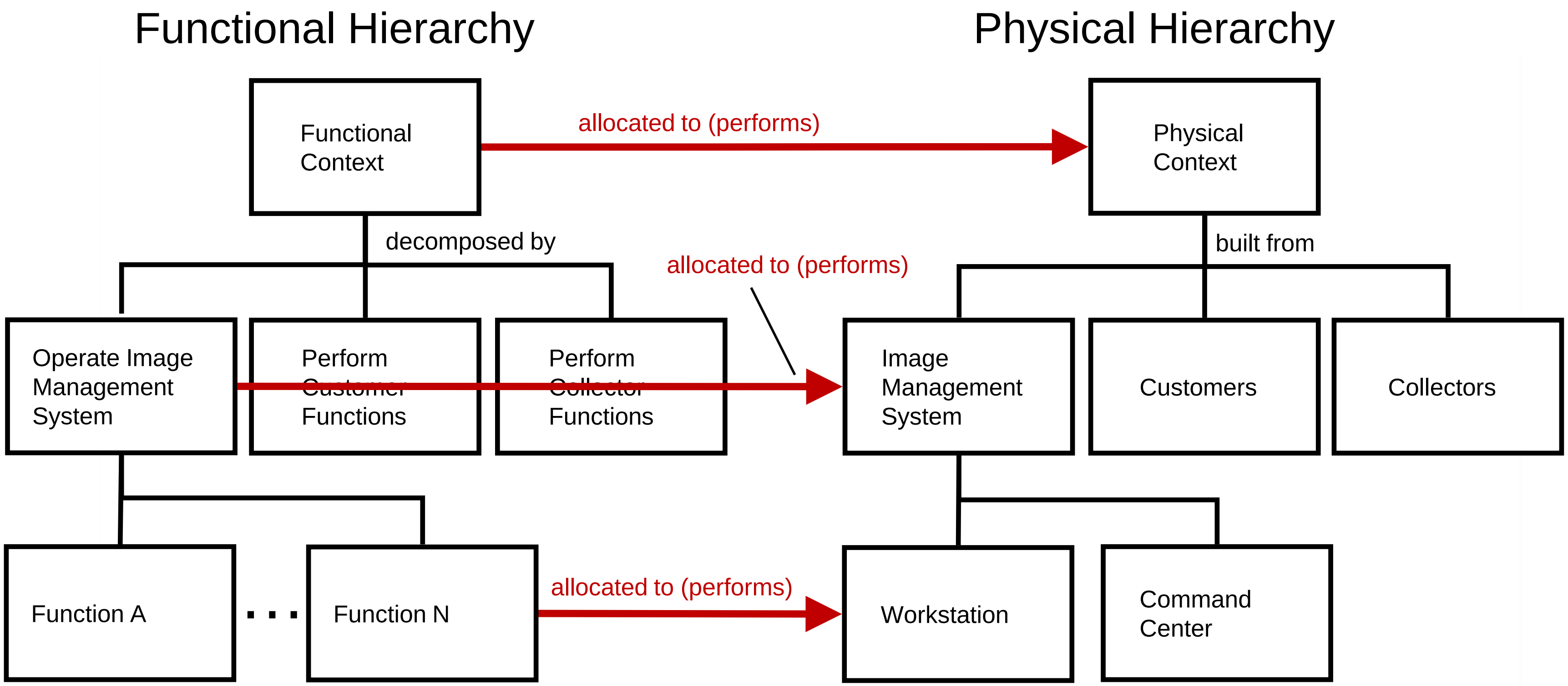


System's behavior is described as a "black box" to identify conditional process flow and performance

- 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



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

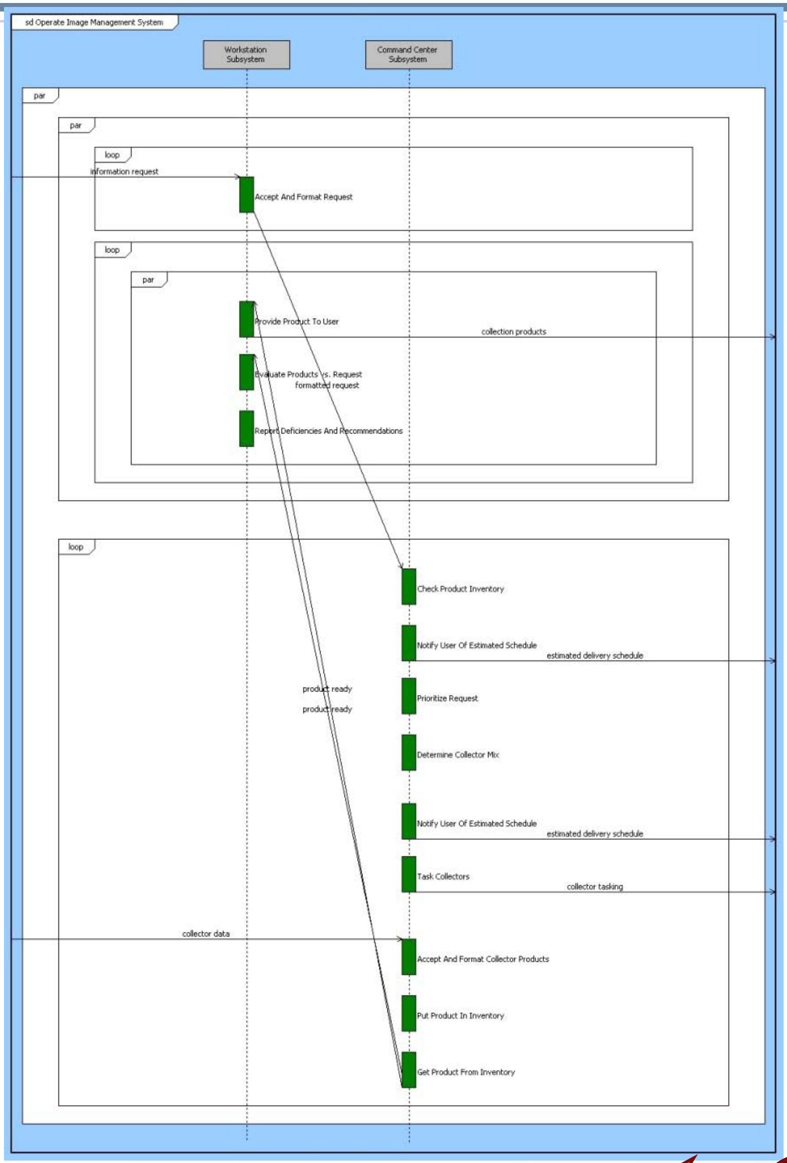
Ref.: DCDS Documentation

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

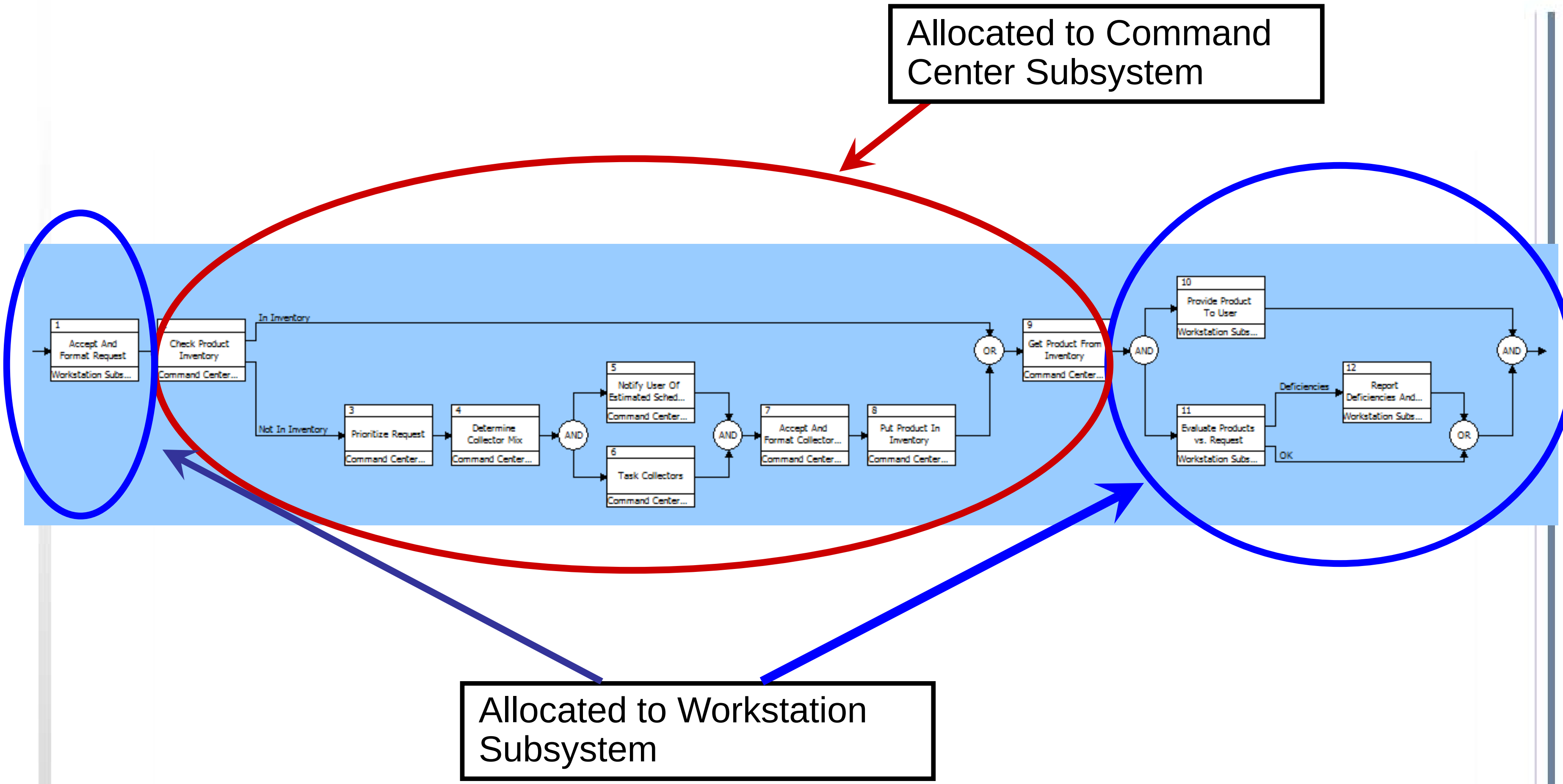
Ref.: DCDS Documentation

Analyzing Message Flows and Sequencing between Components



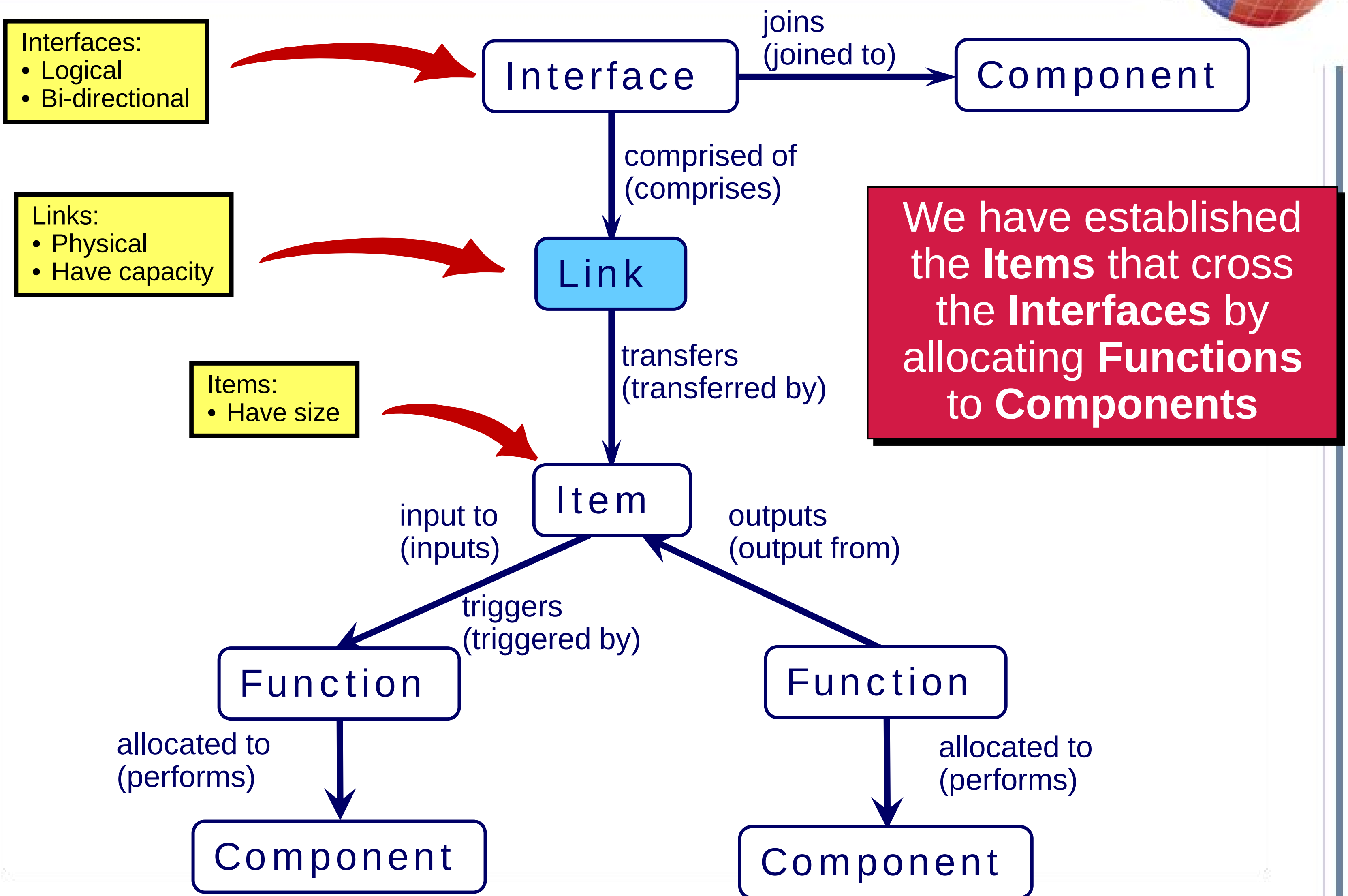
Sequence diagrams focus on triggering information between activities on lifelines to help you understand the way the activity interacts with the greater system.

Allocating the System Functions

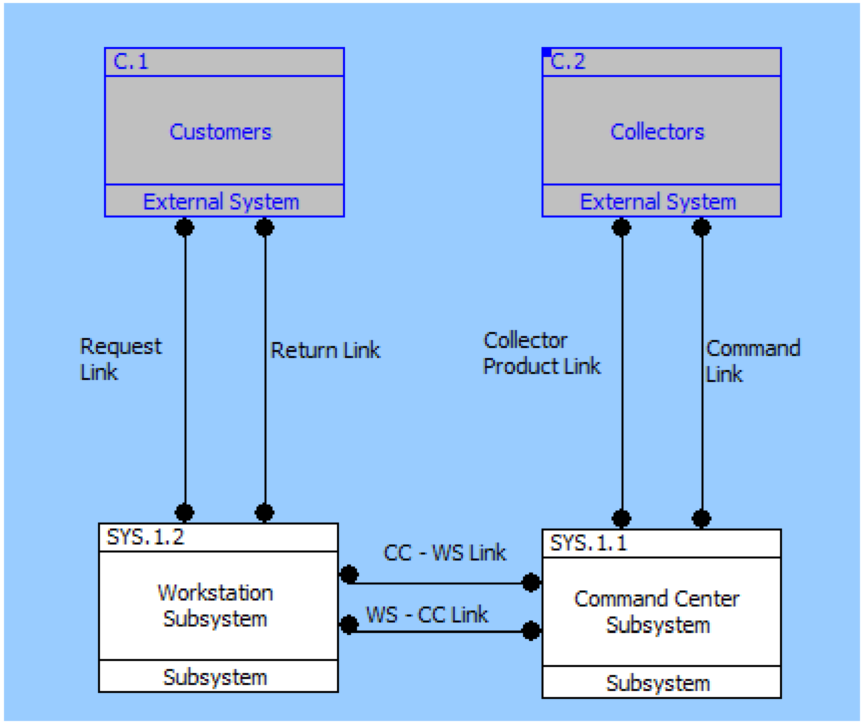


The **Components** will perform the **Functions** that are allocated to them.

Capturing the Interfaces



System Physical Block Diagram



Failure Modes and Effects Analysis

For each component and interface determine:

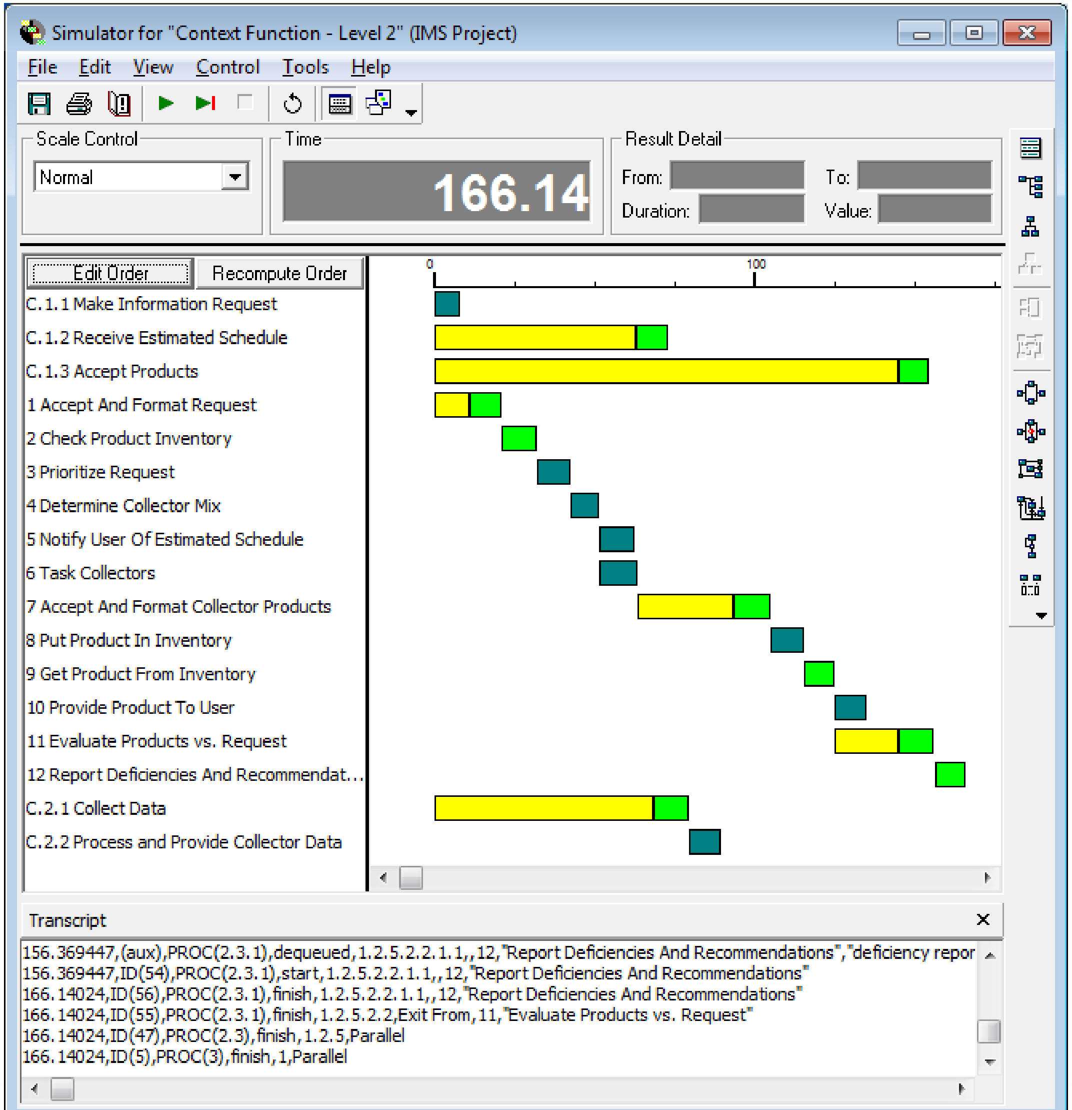
- Requirements and performance indices
- Potential failure modes
  - Probability of each
  - Avoidance approaches
  - Detection strategies
  - Recovery strategies
    - Feasible to continue in spite of failure?
    - Feasible to resume normal behavior after replacement, repair, or end of intermittent 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

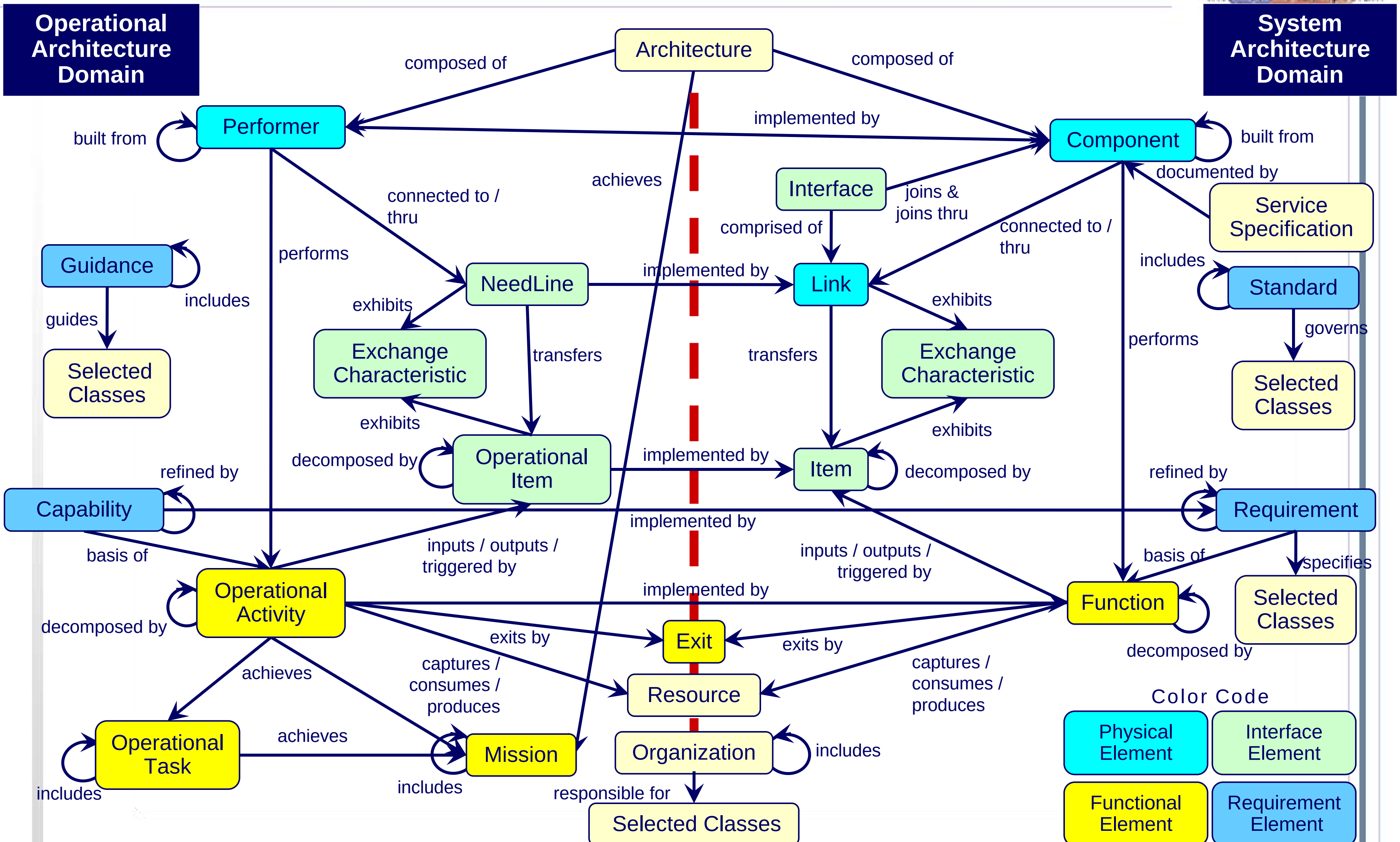
The Discrete Event Simulator Supports Analysis and Design at All Levels

- 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



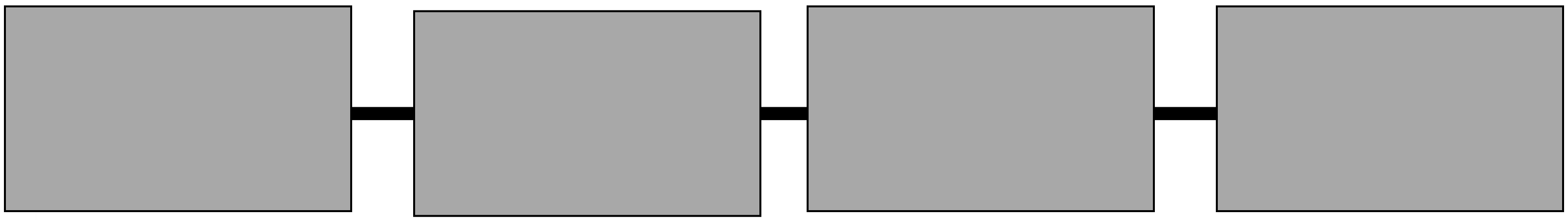
Architectures and DoDAF – A More Complete Schematic



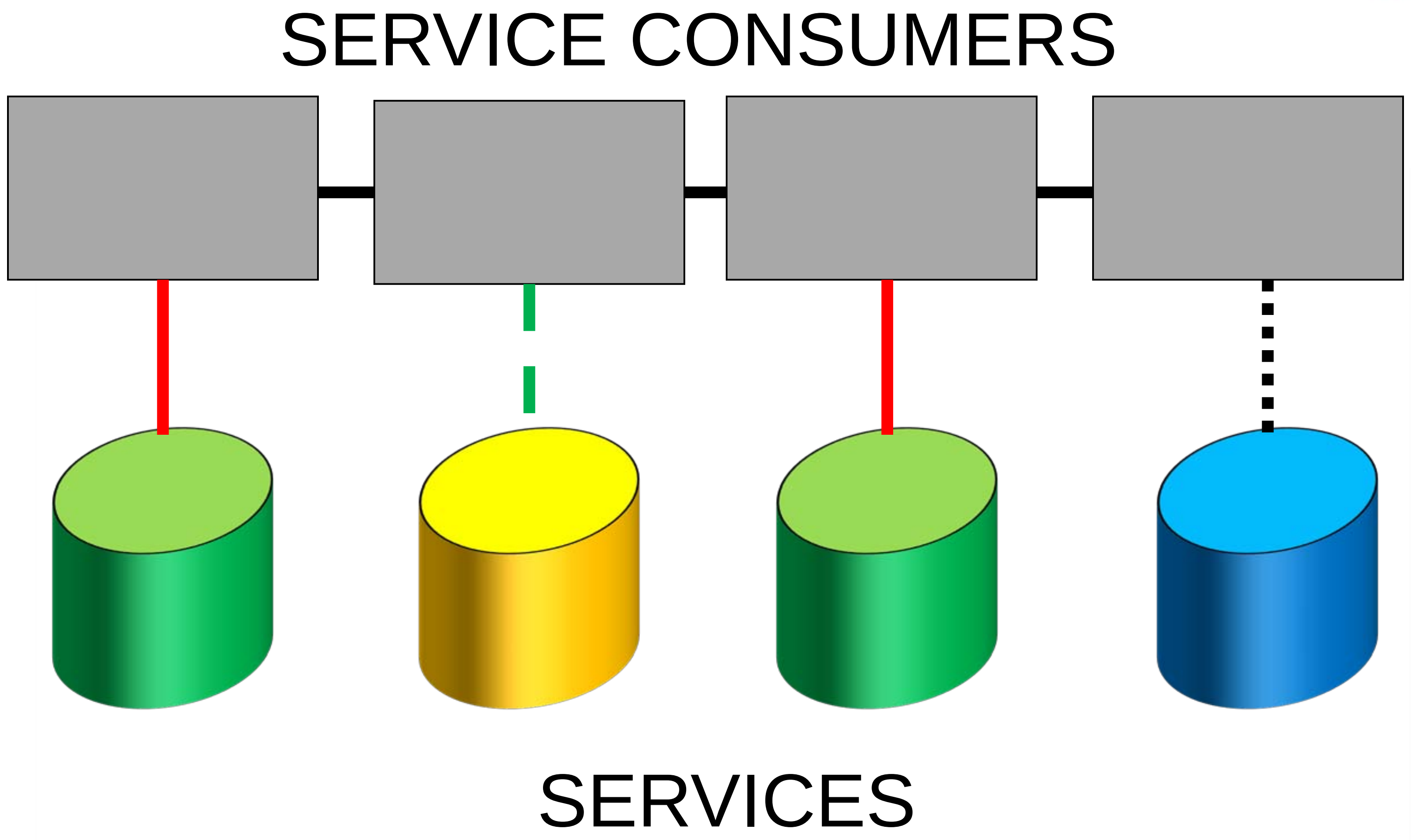
Service Oriented Architecture (SOA)

Pre-SOA Configuration

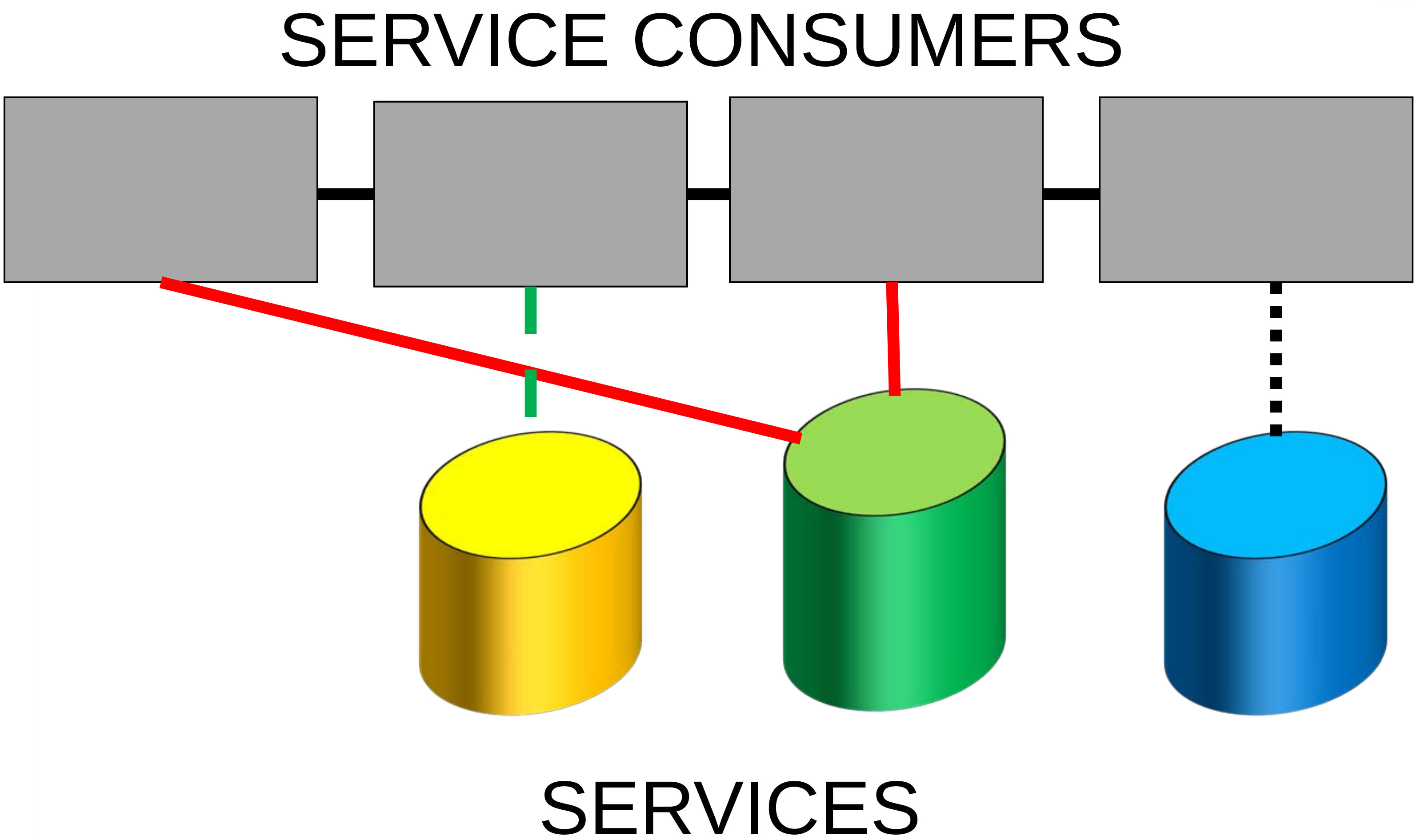
PROCESS

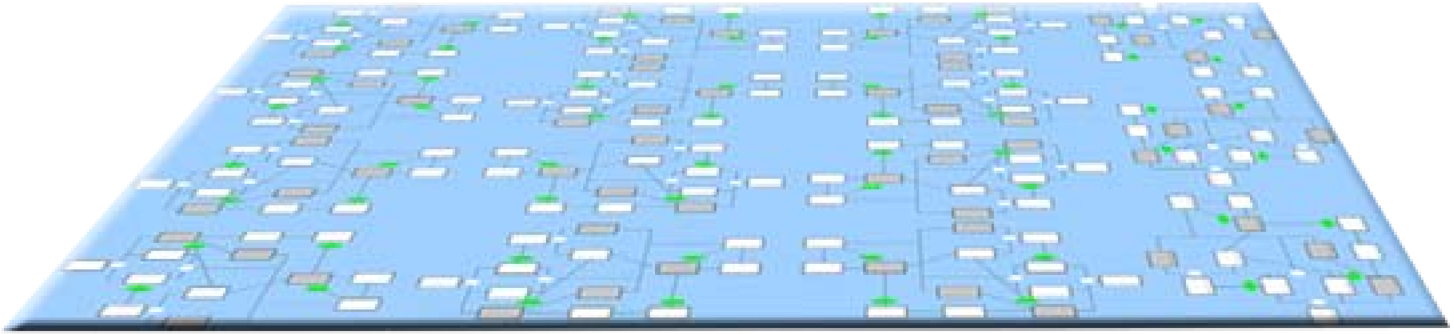


Pre-SOA Configuration



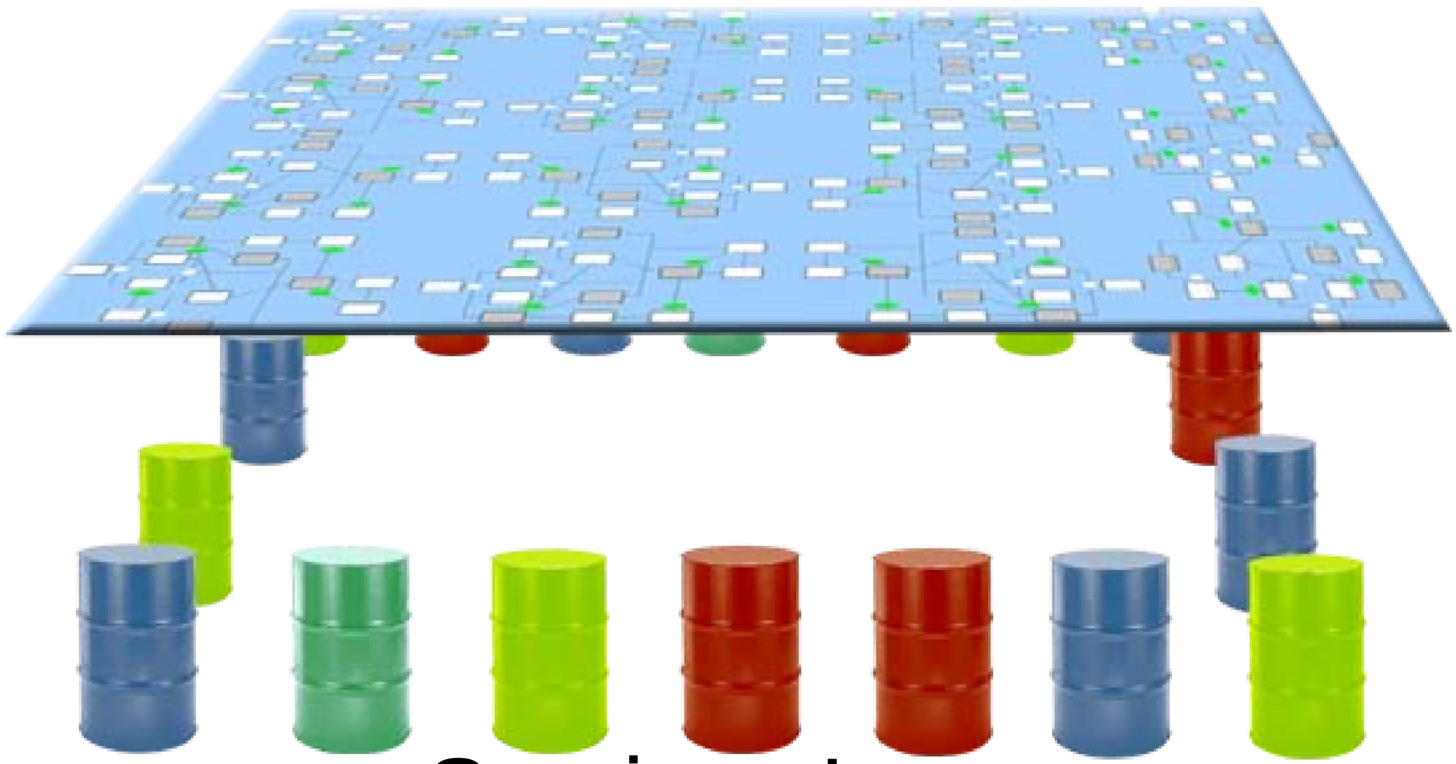
Pre-SOA Configuration





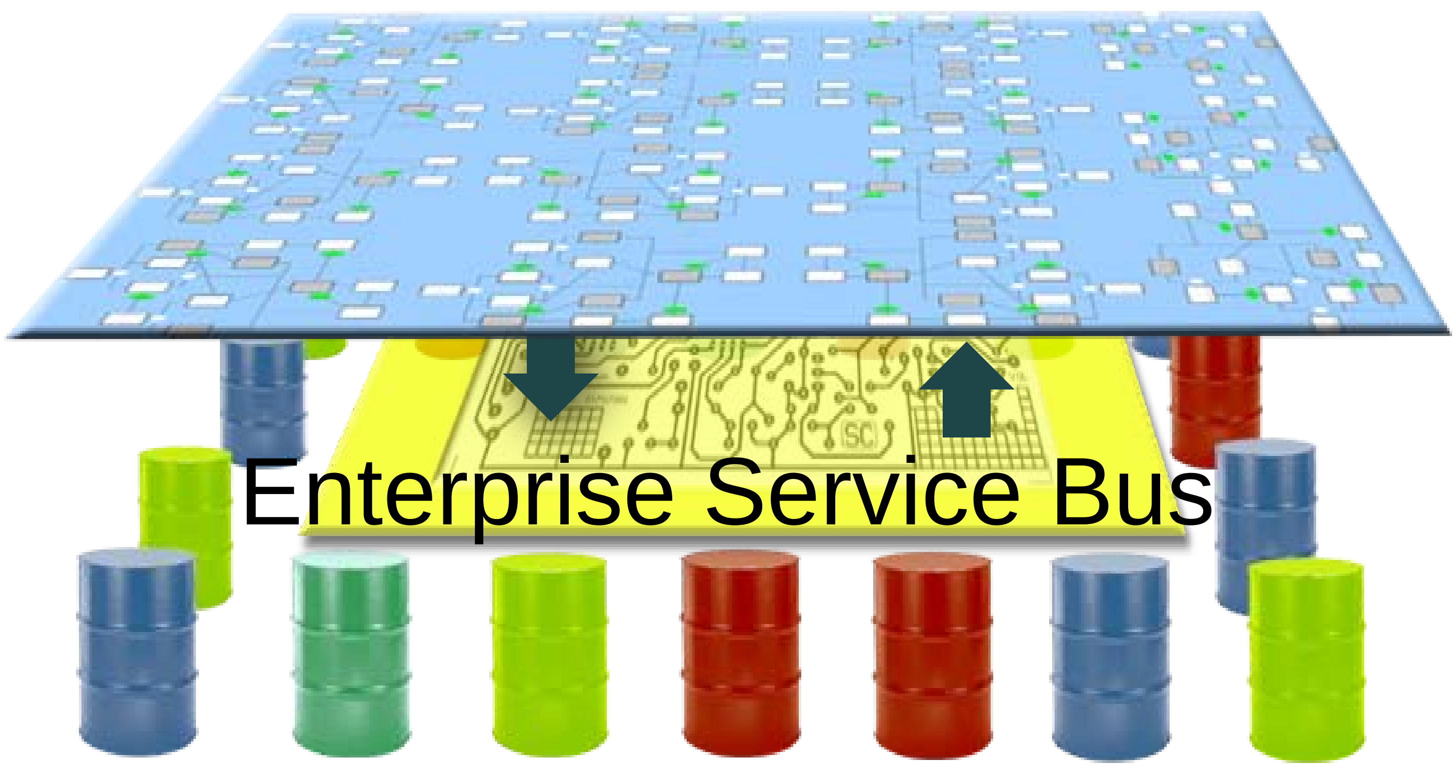
Process (Orchestration) Layer

SOA Basics

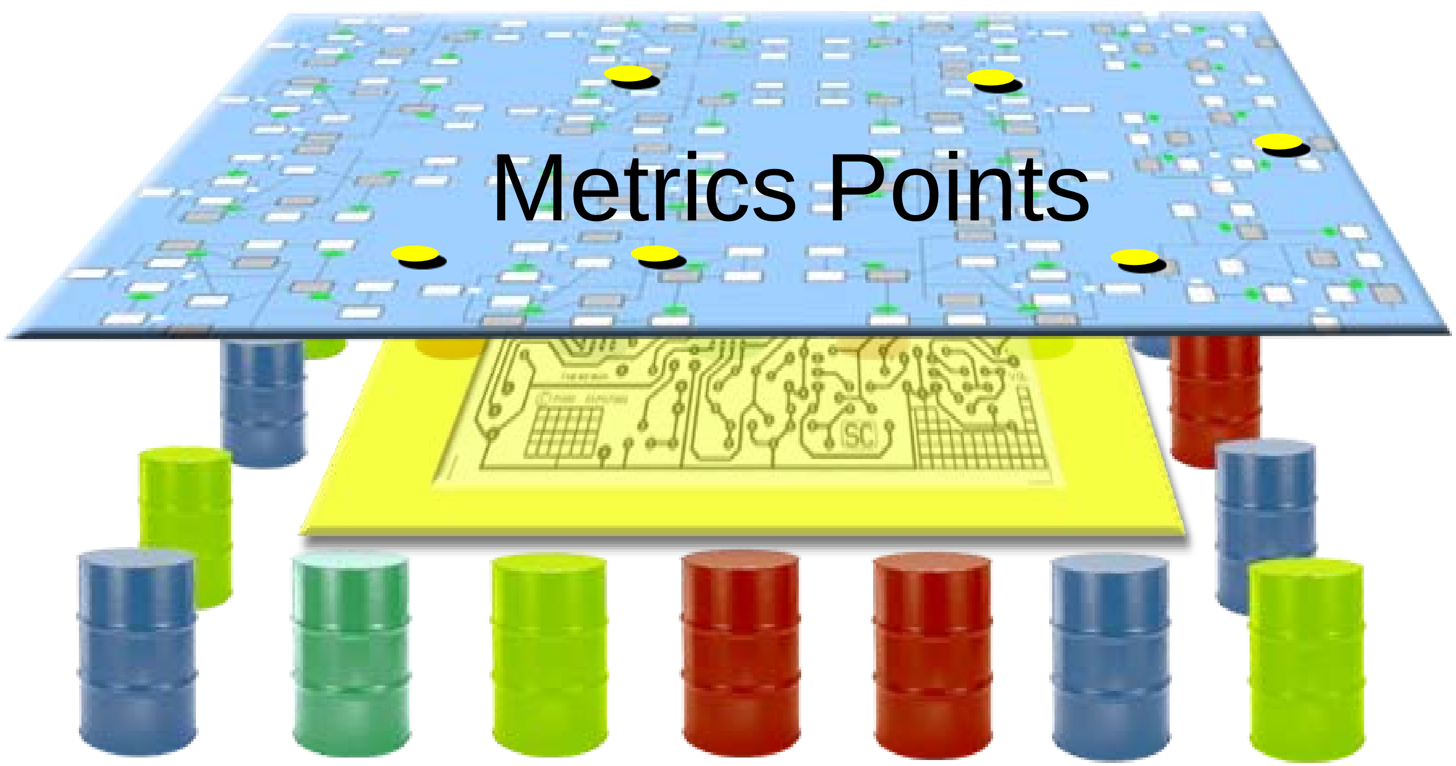


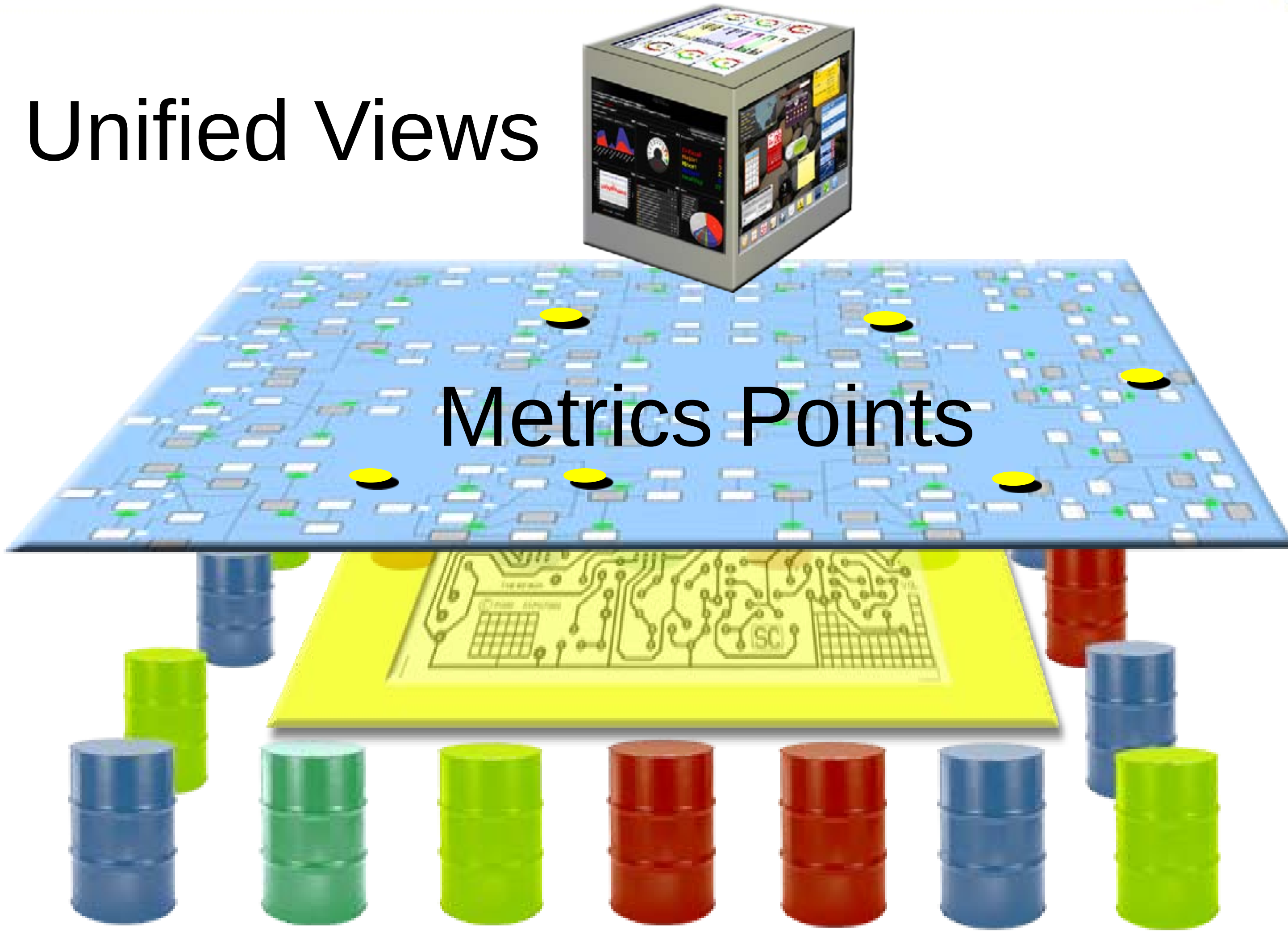
Services Layer

SOA Basics



SOA Basics




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.

DoD Architecture Framework, Version 1.0, (09 February 2004) Volume 1, p. 1-5

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

Eoin Woods

Architectures in Context

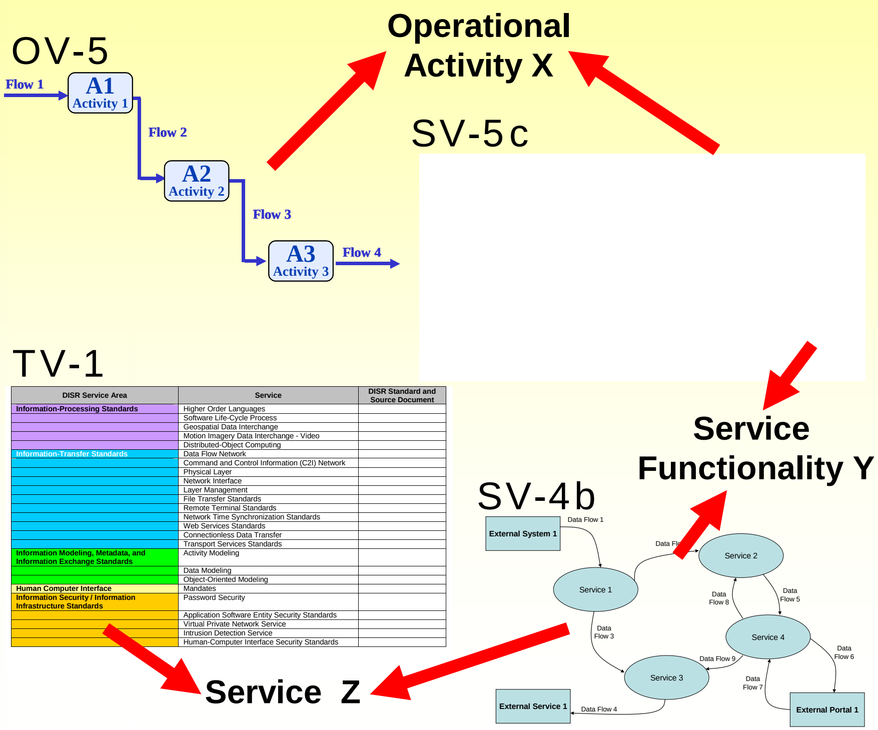
Level	Scope	Detail	Impact	Audience
Enterprise Architecture	Agency/ Organization	Low	Strategic Outcomes	All Stakeholders
Segment Architecture	Line of Business	Medium	Business Outcomes	Business Owners
Solution Architecture	Function/ Process	High	Operational Outcomes	Users and Developers

Reprinted from *2006 Federal Enterprise Architecture Practice Guidance*, US OMB

Integrated and Federated Architectures

Integrated Architecture

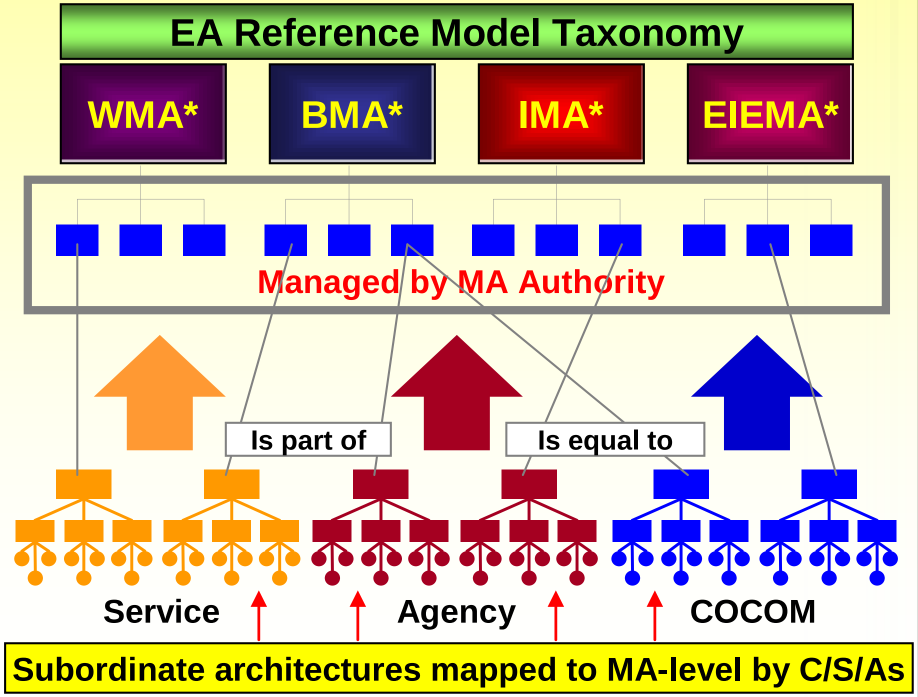
- ✓ An architecture where architecture data elements are uniquely identified and consistently used across all products and views within the architecture.



DISR Service Area	Service	DISR Standard and Source Document
Information-Processing Standards	Higher Order Languages	
	Software Life-Cycle Process	
	Geospatial Data Interchange	
	Motion Imagery Data Interchange - Video	
	Distributed-Object Computing	
Information-Transfer Standards	Data Flow Network	
	Command and Control Information (C2I) Network	
	Physical Layer	
	Network Interface	
	Layer Management	
	File Transfer Standards	
	Remote Terminal Standards	
	Network Time Synchronization Standards	
	Web Services Standards	
	Connectionless Data Transfer	
	Transport Services Standards	
Information Modeling, Metadata, and Information Exchange Standards	Activity Modeling	
	Data Modeling	
	Object-Oriented Modeling	
Human Computer Interface	Mandates	
Information Security / Information Infrastructure Standards	Password Security	
	Application Software Entity Security Standards	
	Virtual Private Network Service	
	Intrusion Detection Service	
	Human-Computer Interface Security Standards	

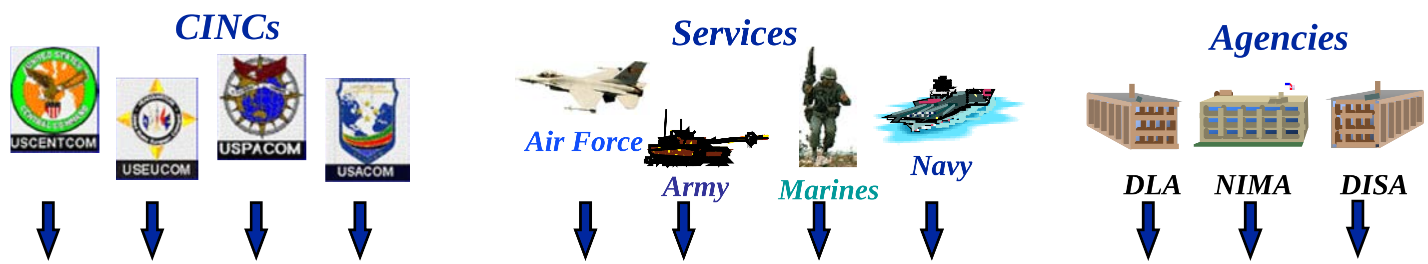
Federated Architecture

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

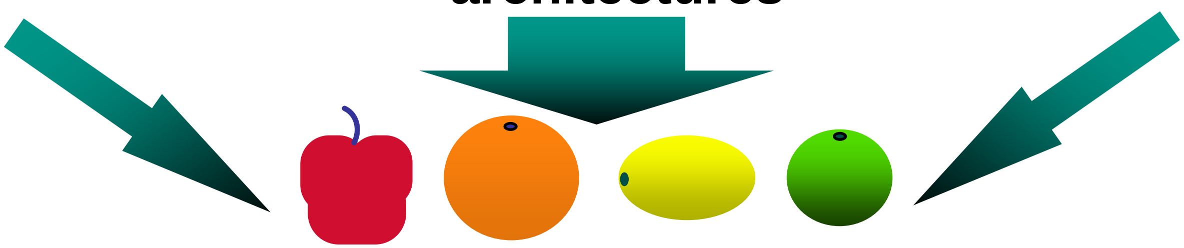


From Architectures to a Framework: Why is a Framework Needed?

Organizations are developing major systems that need to interface and interact



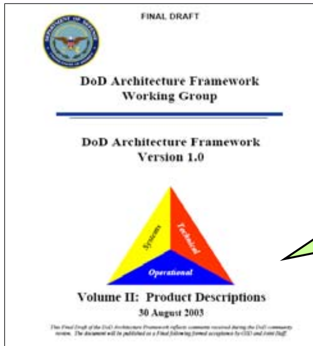
Differences in content and formats inhibit comparison of architectures



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

Reprinted from "C4ISR INCOSE Tutorial", A.H. Levis and L.W.Wagenhals, March 2001

DoDAF Evolution To Support "Fit For Purpose" Architecture


FINAL DRAFT

DoD Architecture Framework Working Group

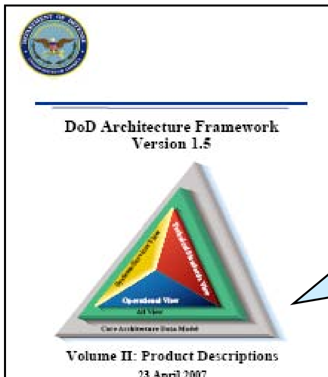
DoD Architecture Framework Version 1.0

Volume II: Product Descriptions

30 August 2003

(Published in 2003)

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

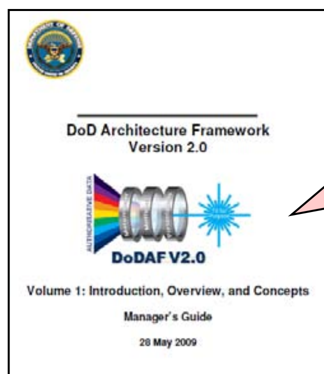

DoD Architecture Framework Version 1.5

Volume II: Product Descriptions

23 April 2007

(Published in 2007)

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


DoD Architecture Framework Version 2.0

Volume 1: Introduction, Overview, and Concepts

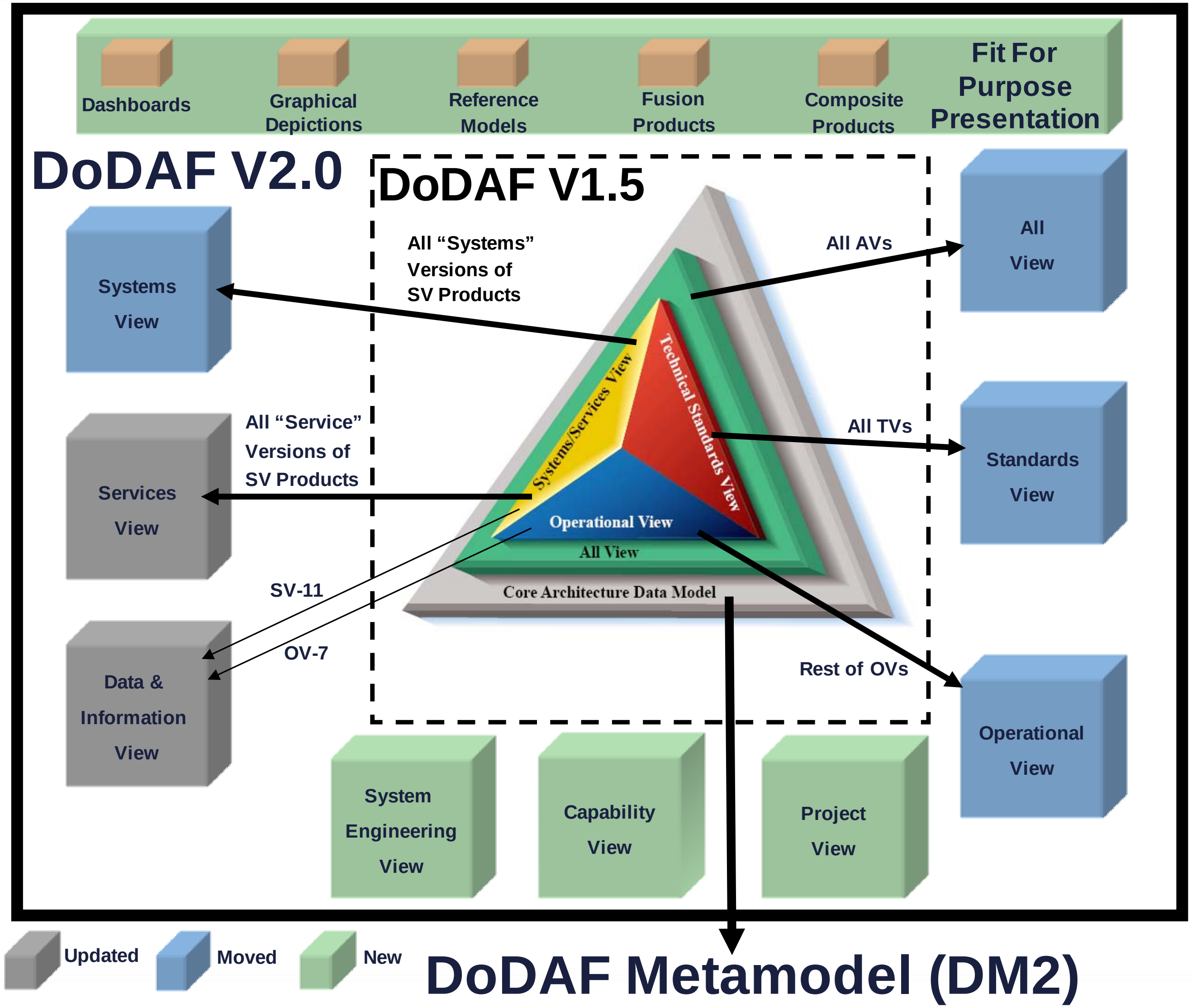
Manager's Guide

28 May 2009

(Published in 2009)

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

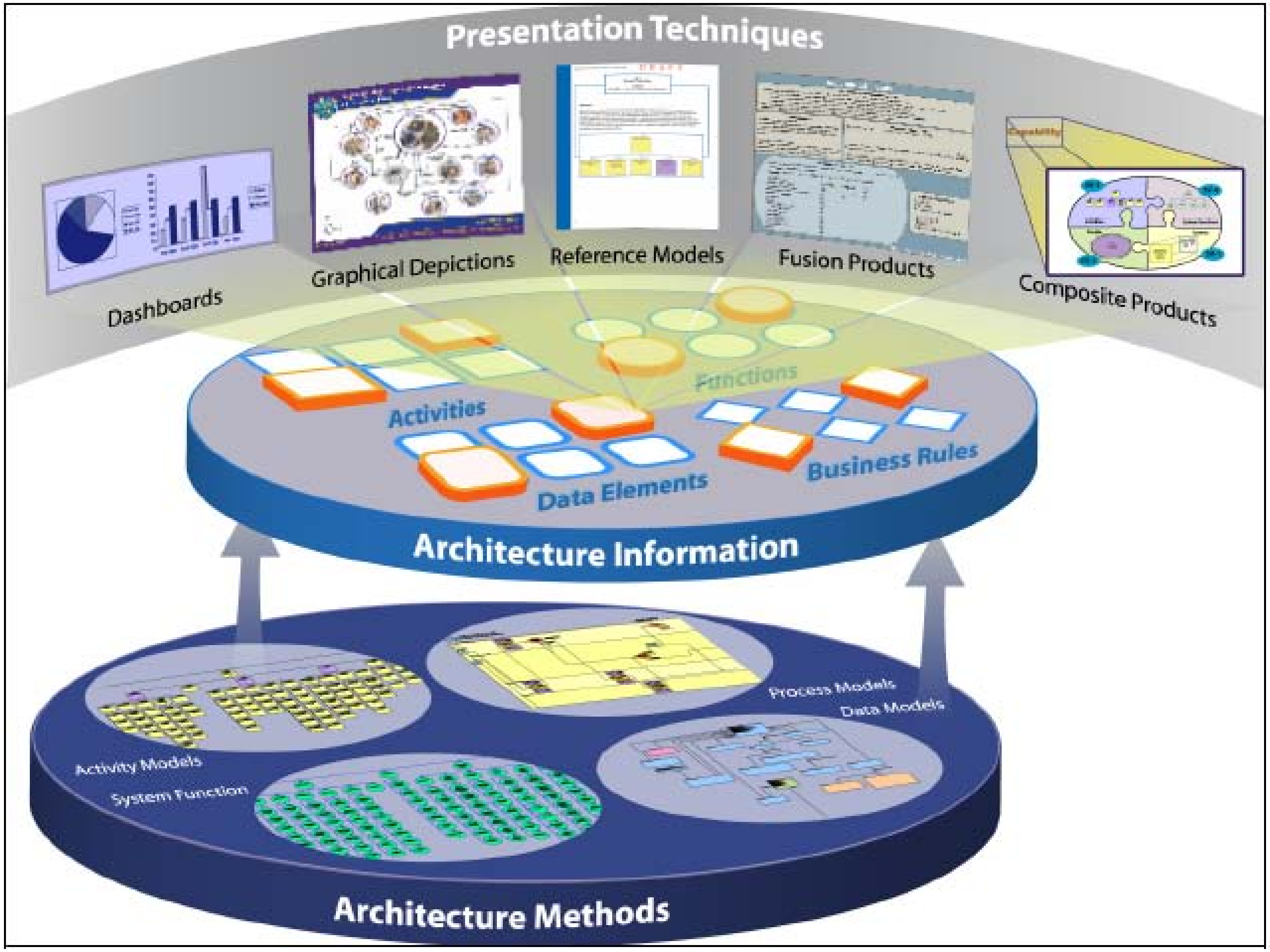
DoDAF 2.0: A Marked Expansion



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

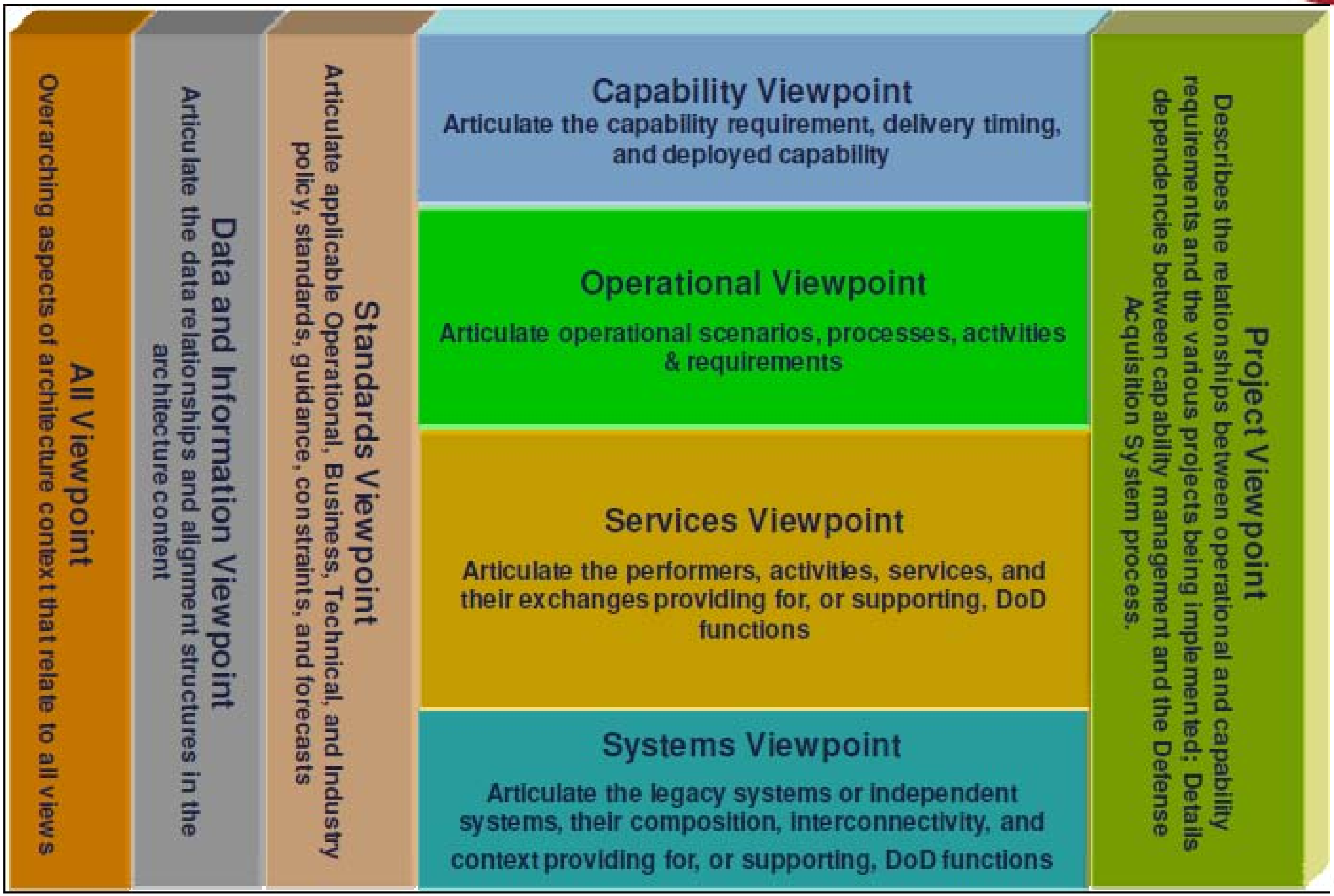
The architect translates the decision-maker's requirements into a set of data that can be used by engineers to design possible solutions.

DoDAF 2.0 Model, View, and Viewpoint Concepts, cont.

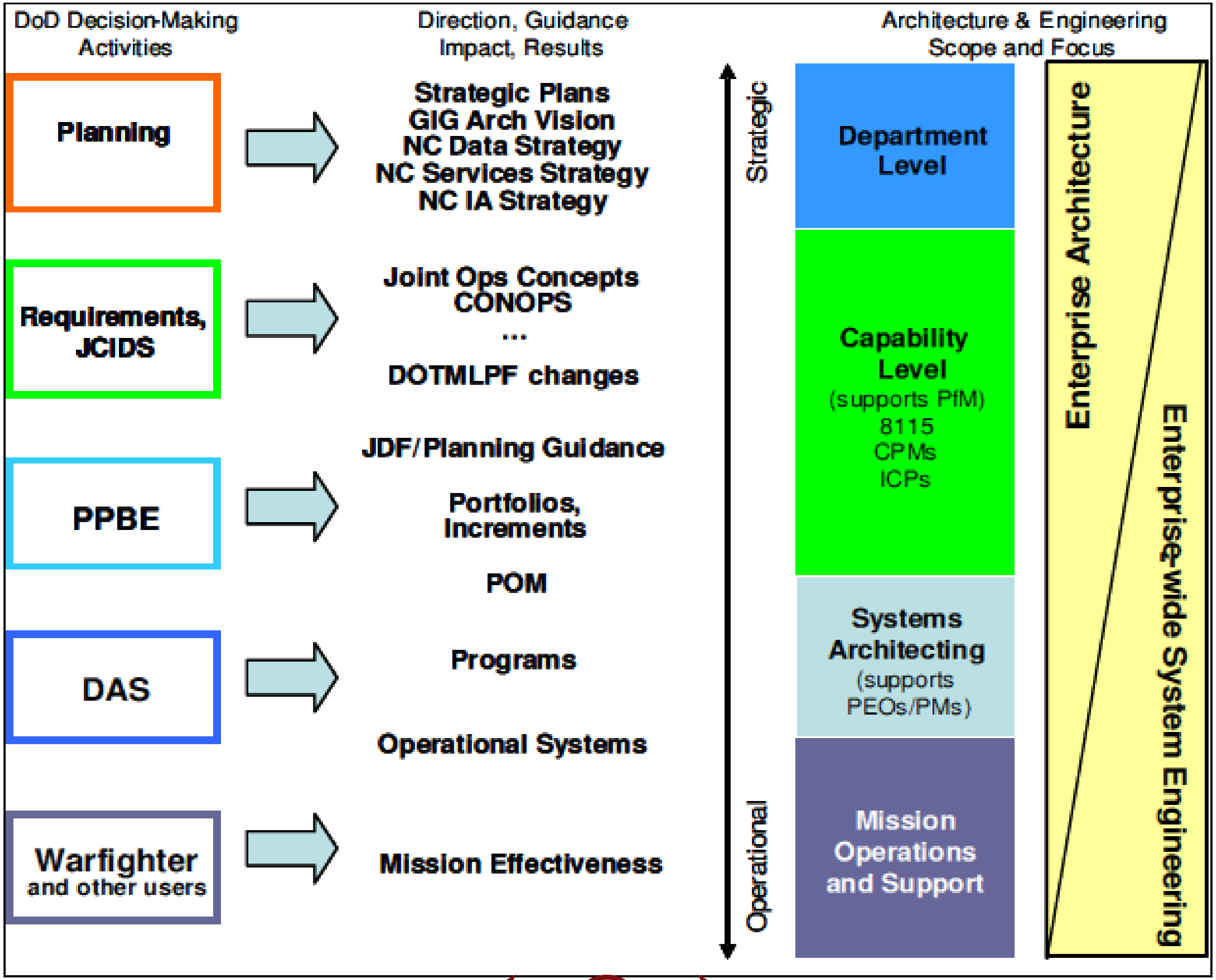
- **Viewpoint** – one or more organizing perspectives for data useful for supporting management decision-making, including
  - The information appearing in individual views
  - How to construct and use the views (by means of an appropriate schema or template)
  - The modeling techniques for expressing 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)

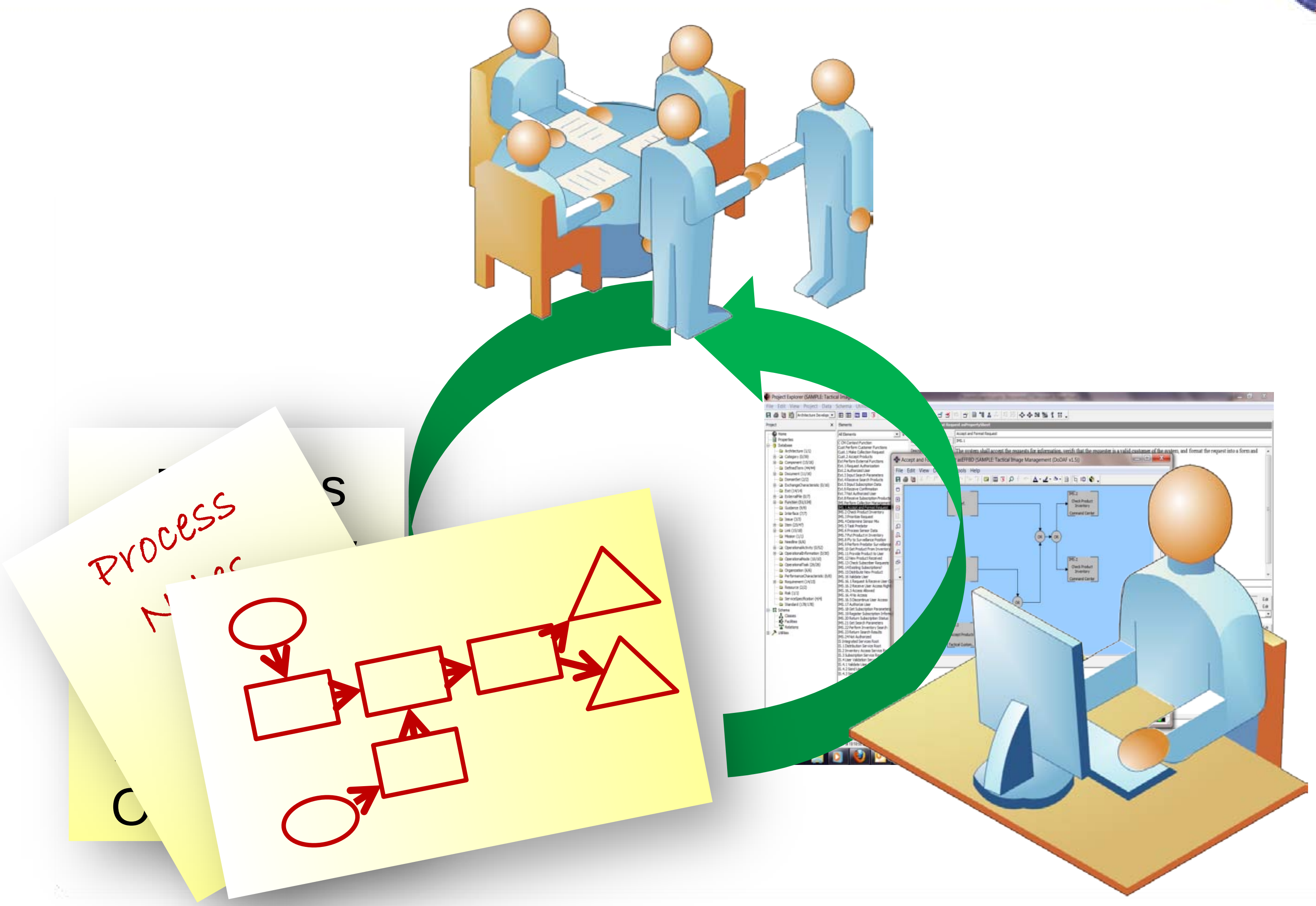
DoDAF 2.0 Viewpoints



Envisioning Architecture Scope



Communication Mismatch: Manager, Architect, and Domain Experts



Challenges in Satisfyting 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

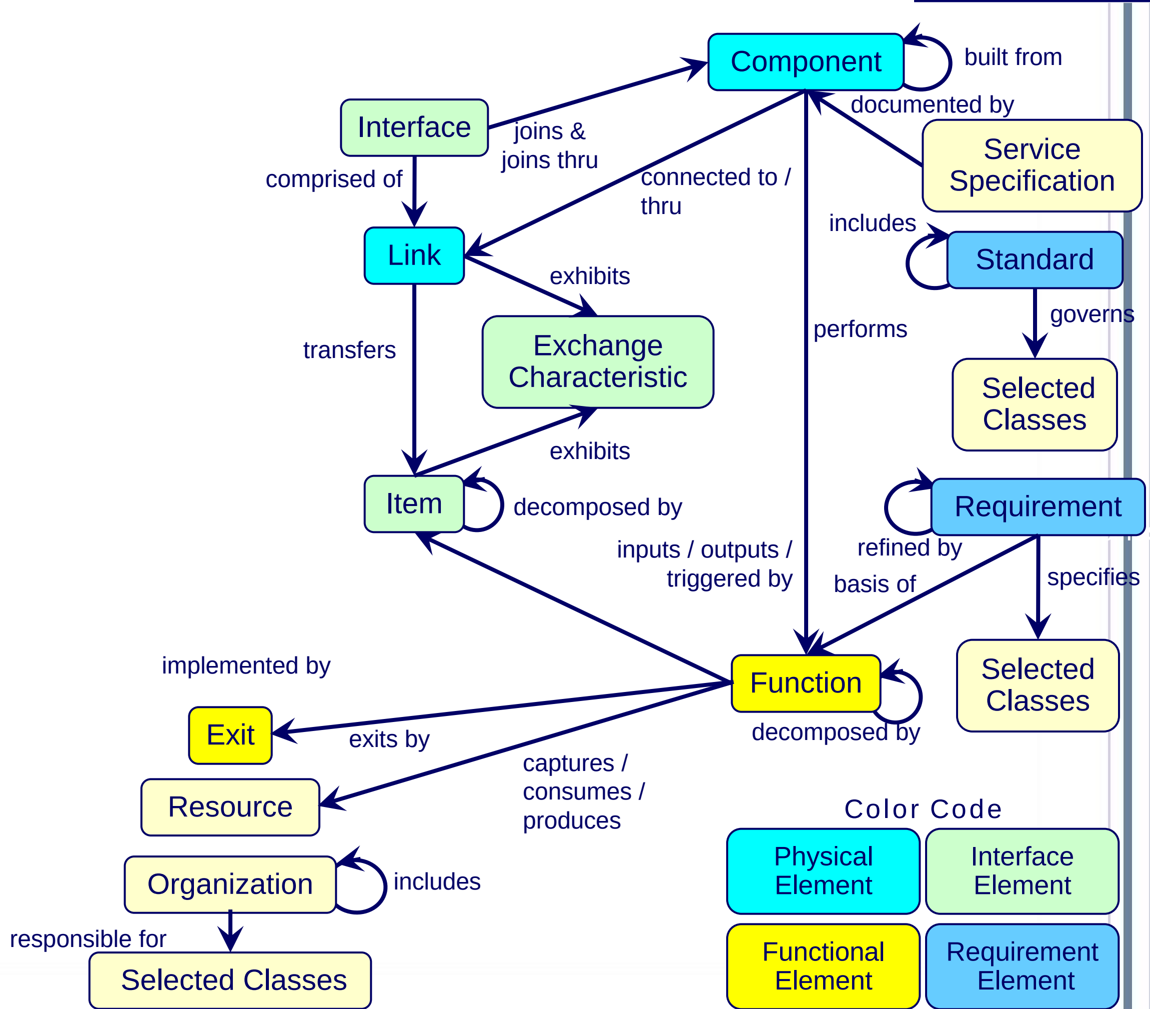
An Integrated Environment Resolves the Data and Semantic Interface Problems

Implementation of integrated DoDAF and system engineering processes provides:

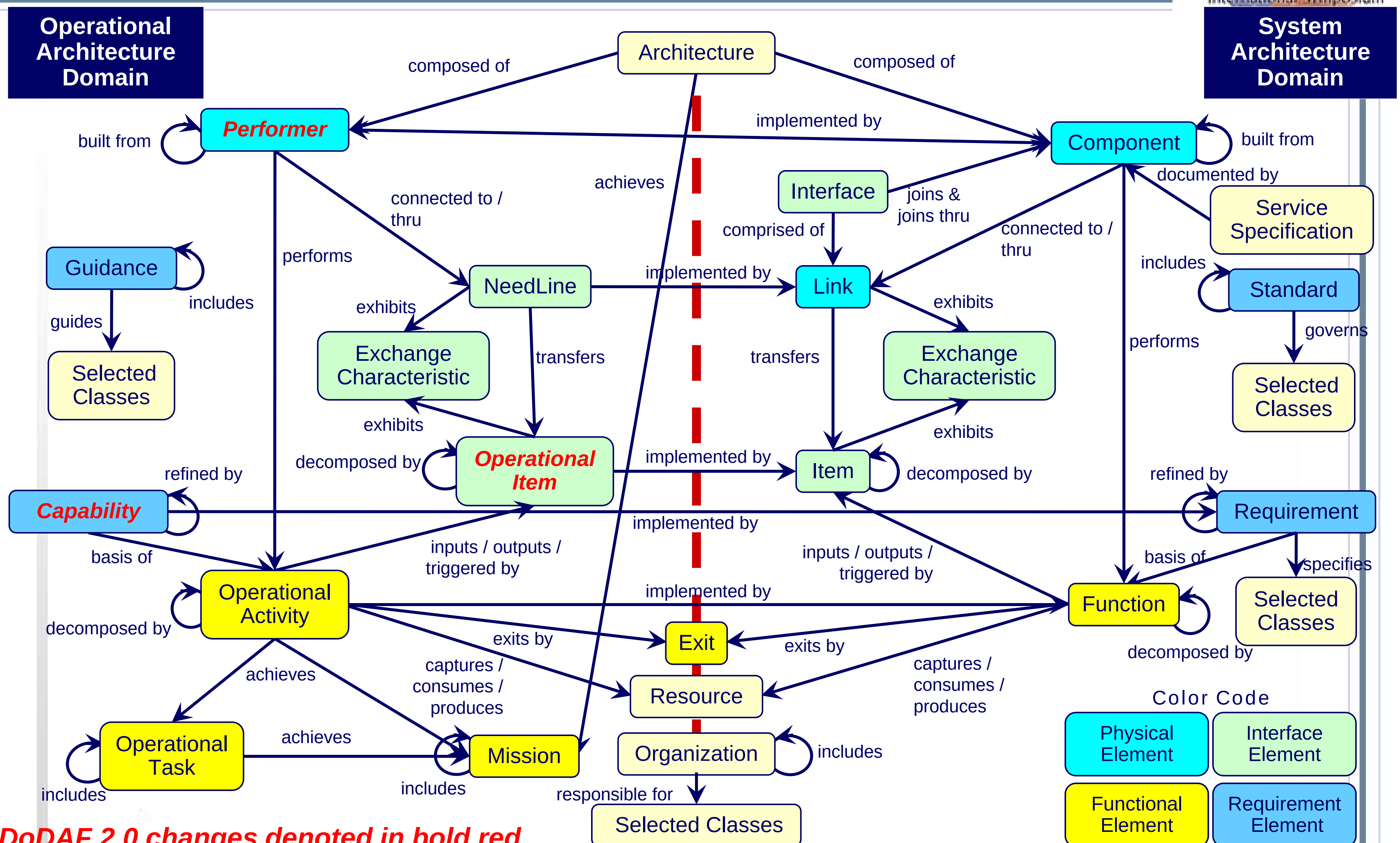
- A repeatable and proven model-based systems engineering methodology
- Integrated models for
  - technical,
  - operational, and
  - system architectures
- A graphical notation to enhance capture and representation for
  - communication and
  - evaluation of candidate architectures
- Executable models (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

A Model-Based Systems Engineering Schema

System Architecture Domain

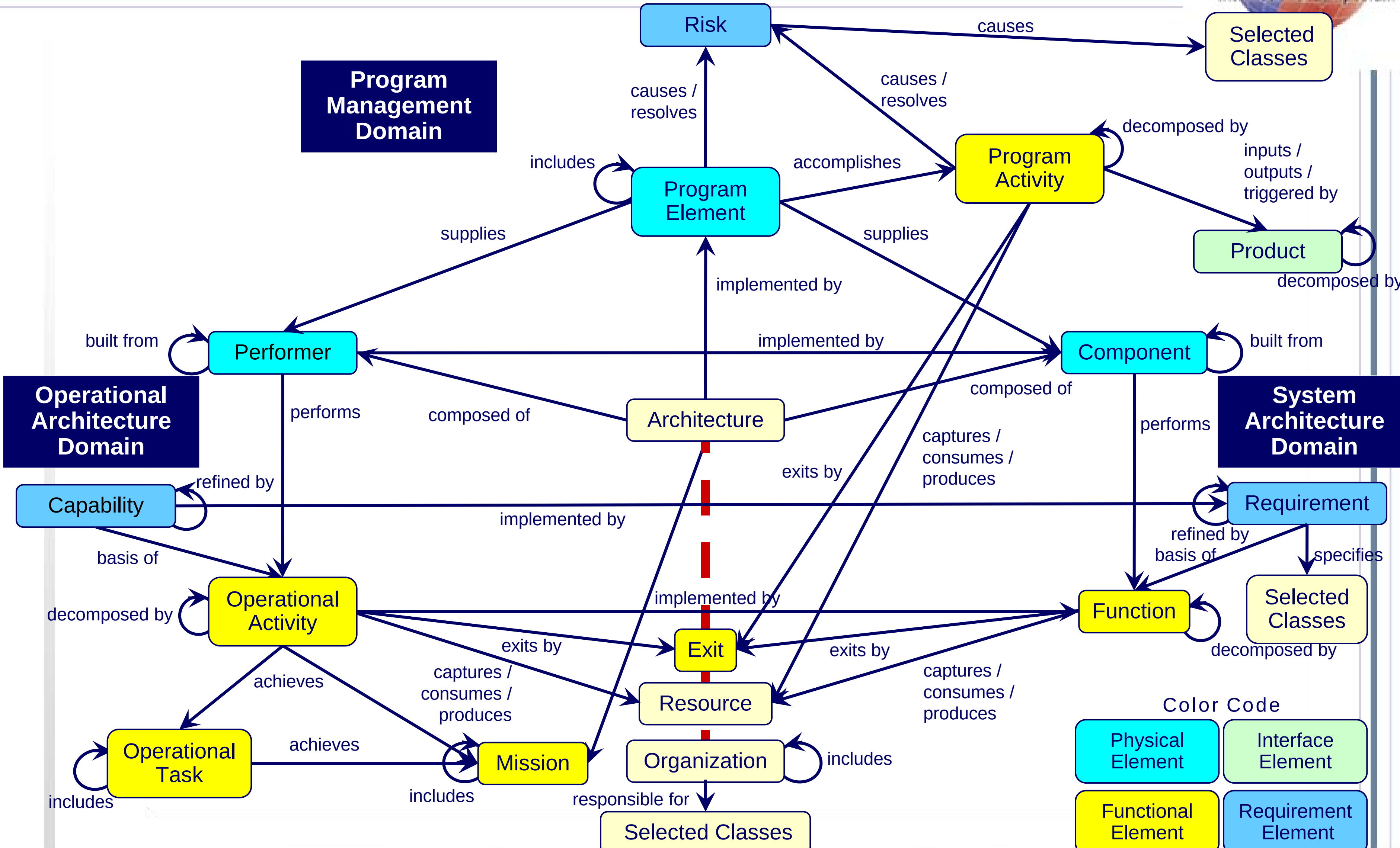


DoDAF 2.0 Schema: System and Operational Architectures



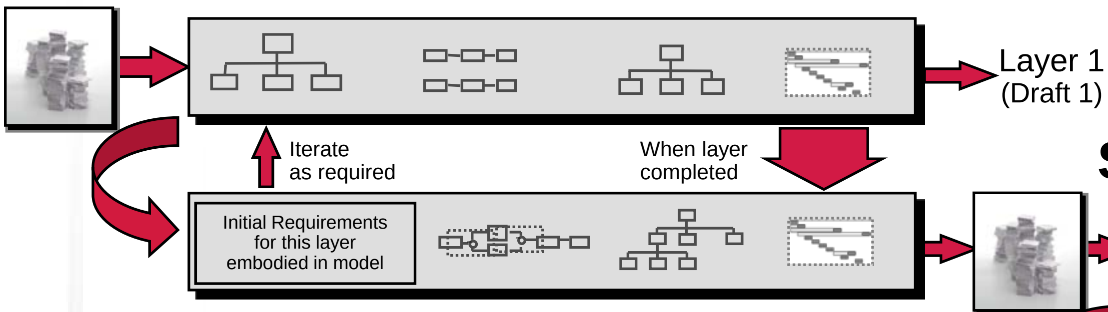
DoDAF 2.0 changes denoted in bold red

DoDAF 2.0 Schema: Capturing the Program Dimension



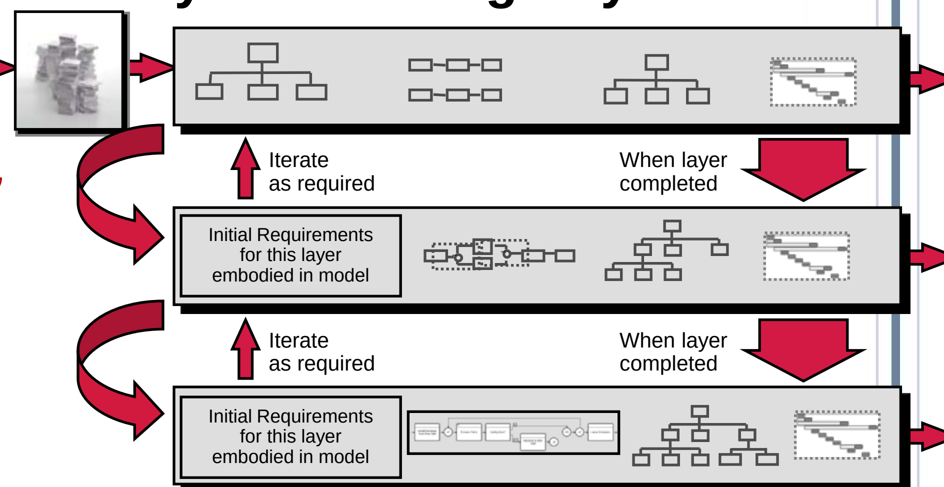
Relationship between OV and SV Generation

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

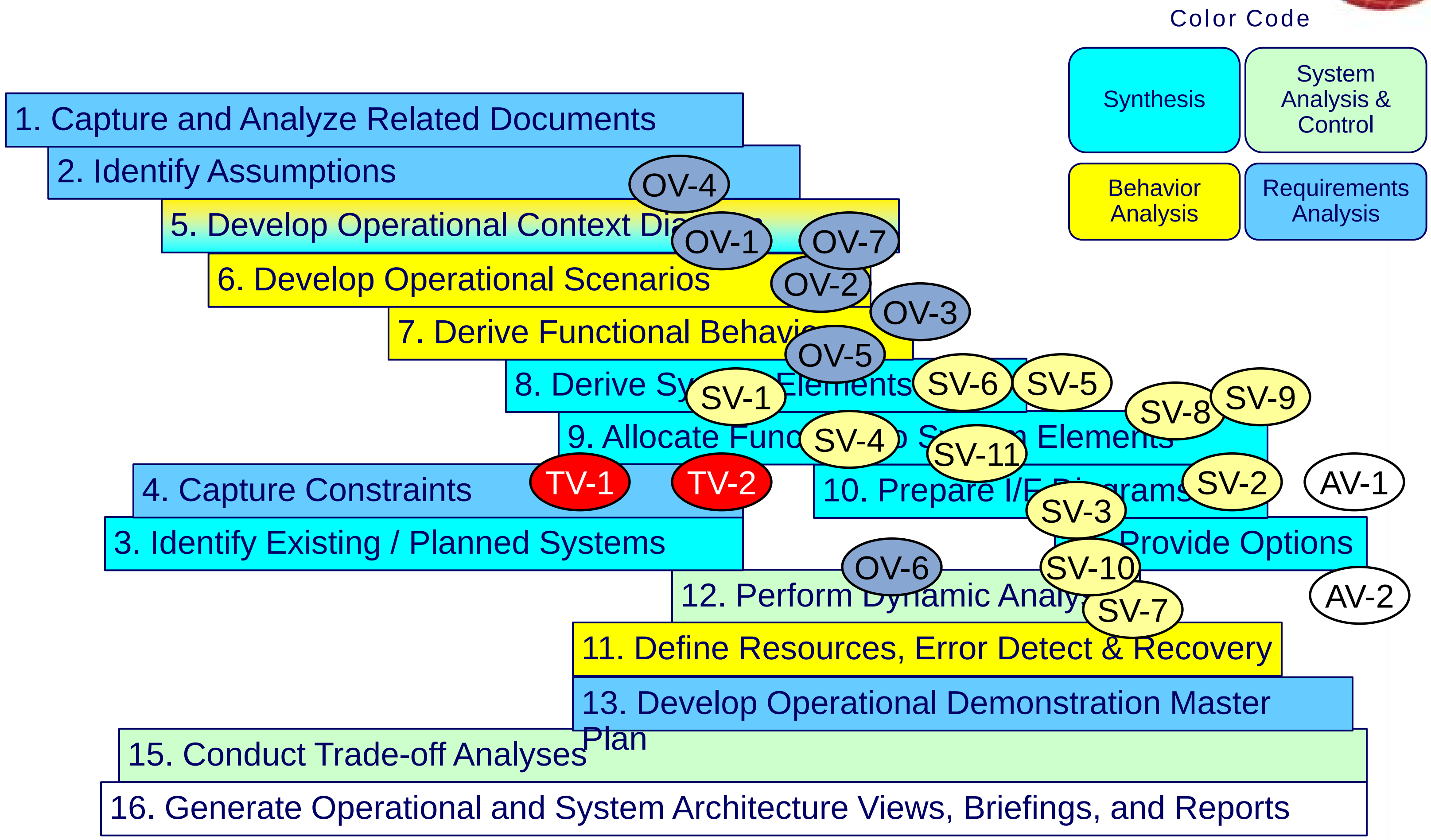
Model-Based Systems Engineering Activities Timeline – Top Down

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.

SCHEDULE

Traditional DoDAF Views within the Systems Development Timeline



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

Defense Acquisition University SoS Research Project Overview, 2003

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 <u>whole that cannot be divided into independent parts</u>

without losing its <u>essential characteristics</u> as a whole.

It follows from this definition that, a system's essential defining properties are the product of the *<u>interactions</u>* 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 *<u>essential properties</u>*.

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

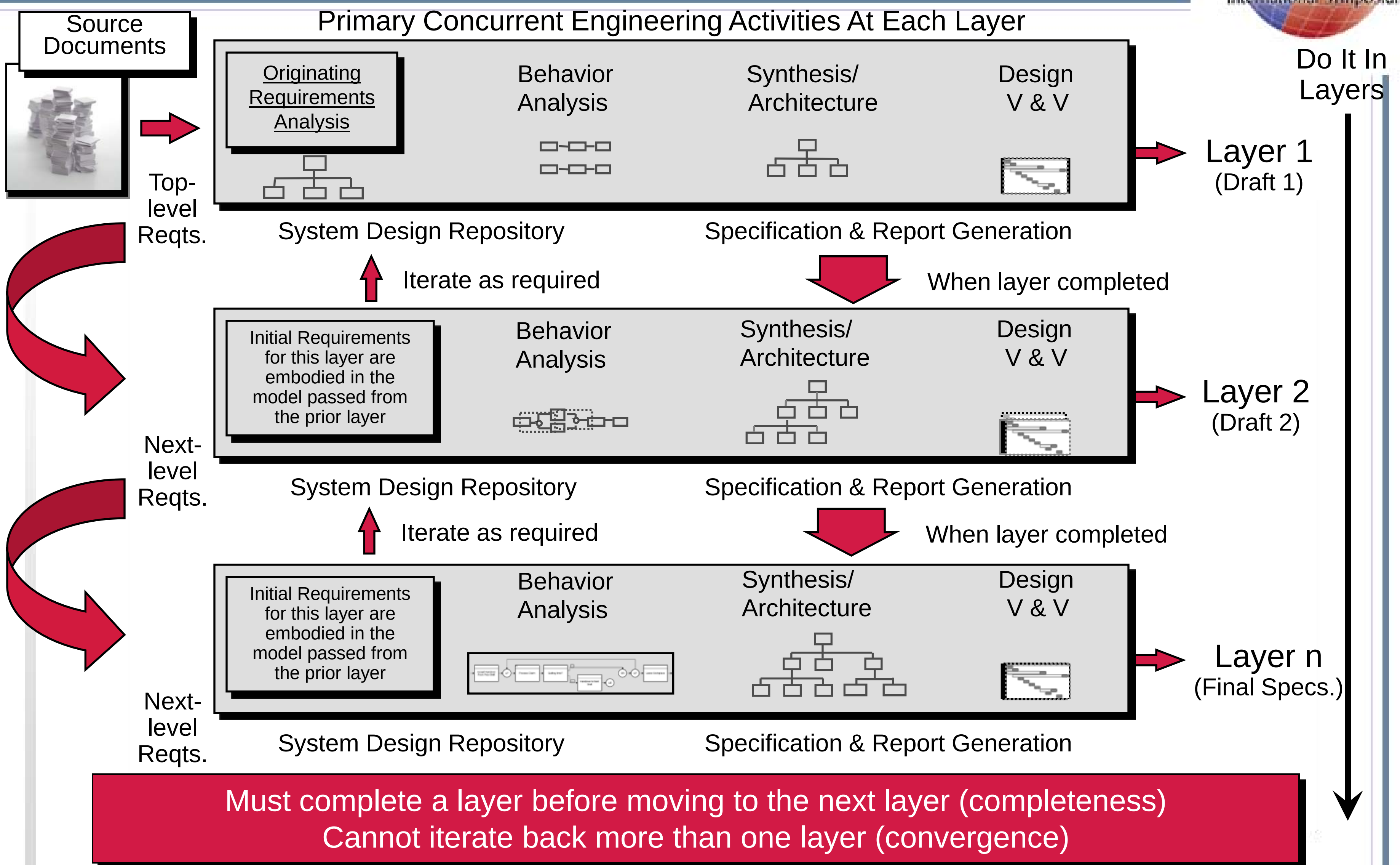
--Russell Ackoff

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.

Source of document: Federation of American Scientists website, www.fas.org

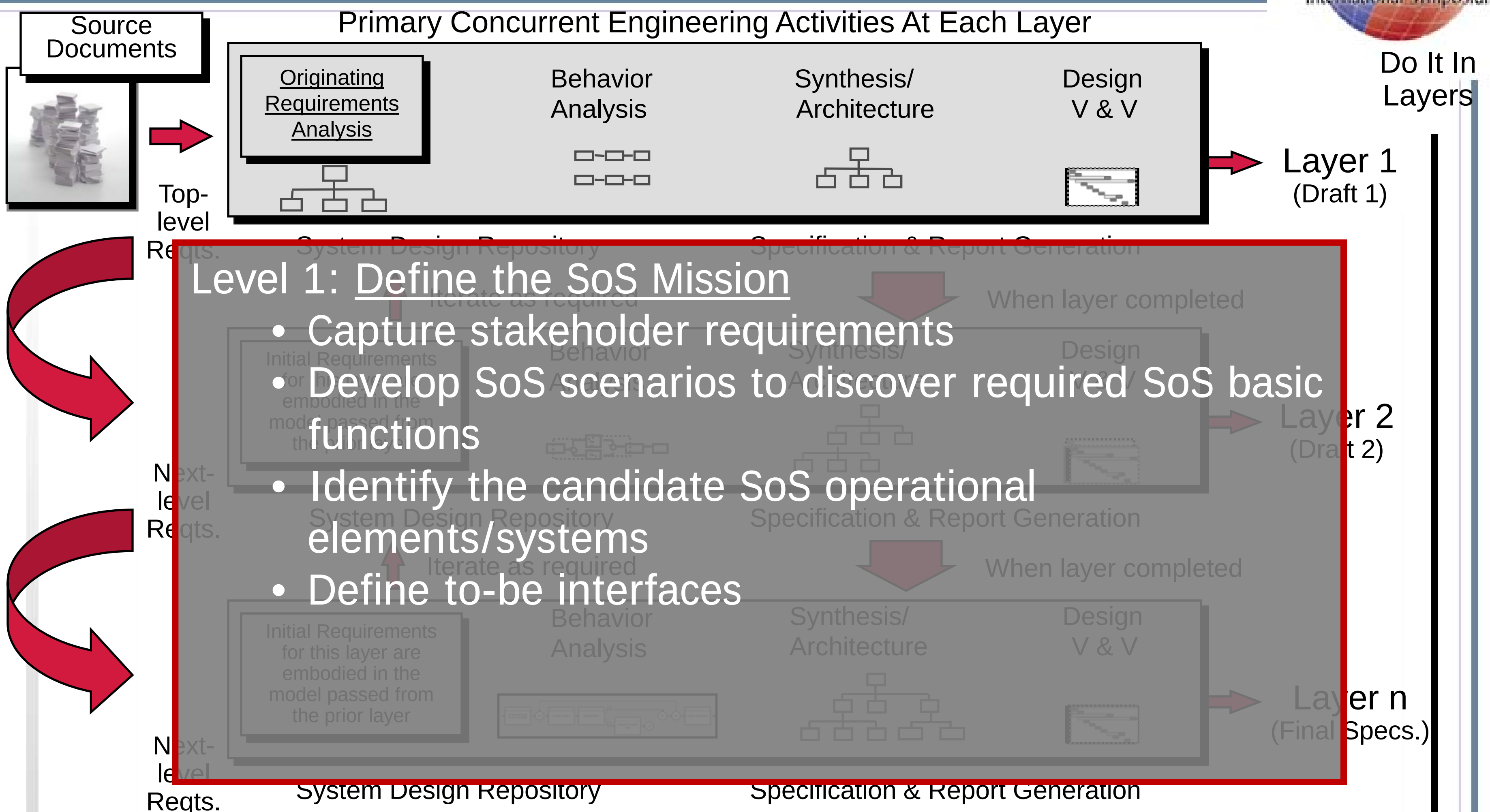
MBSE – The Onion Model for the SoS Example



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

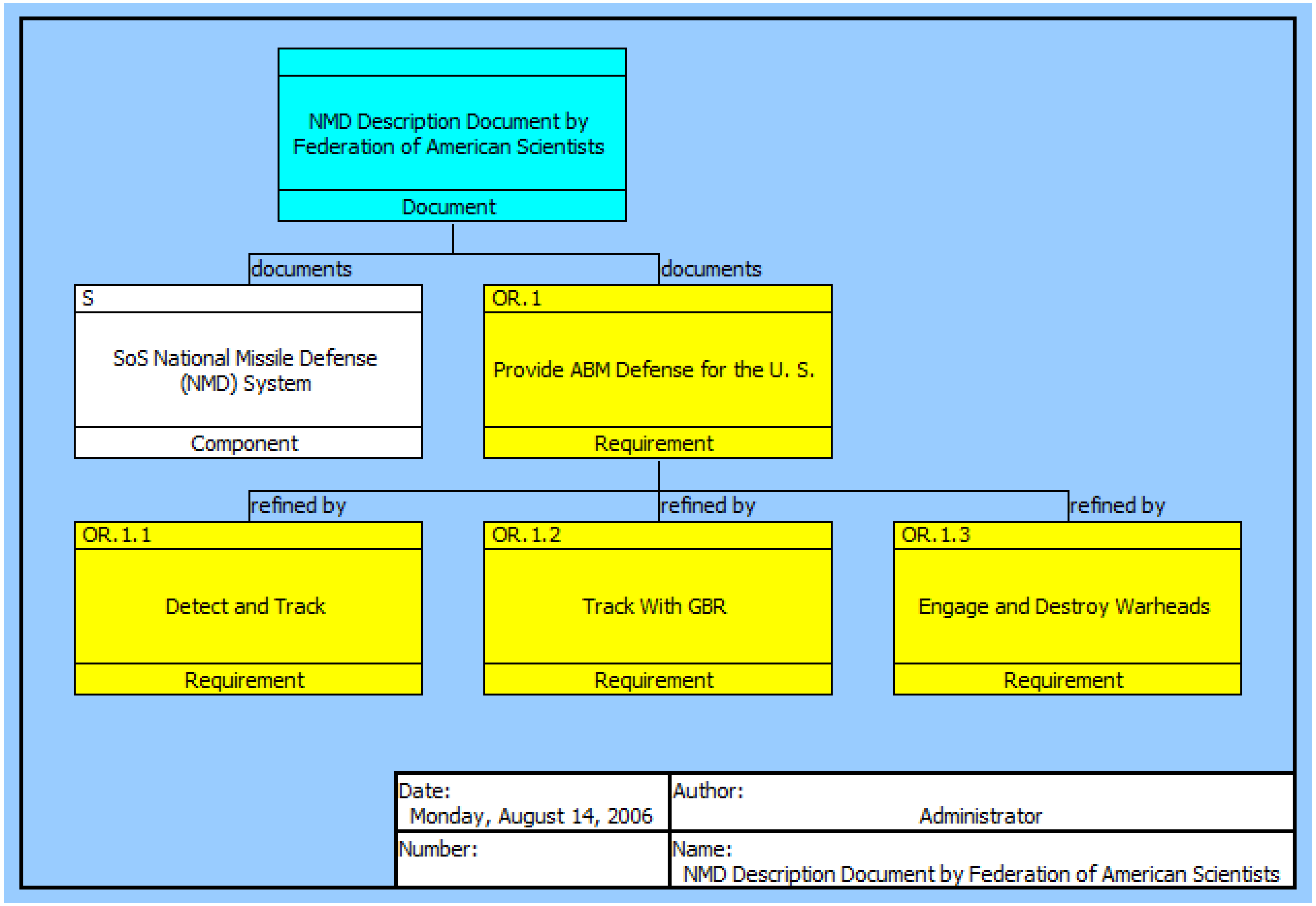


Level 1: Define the SoS Mission

- 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

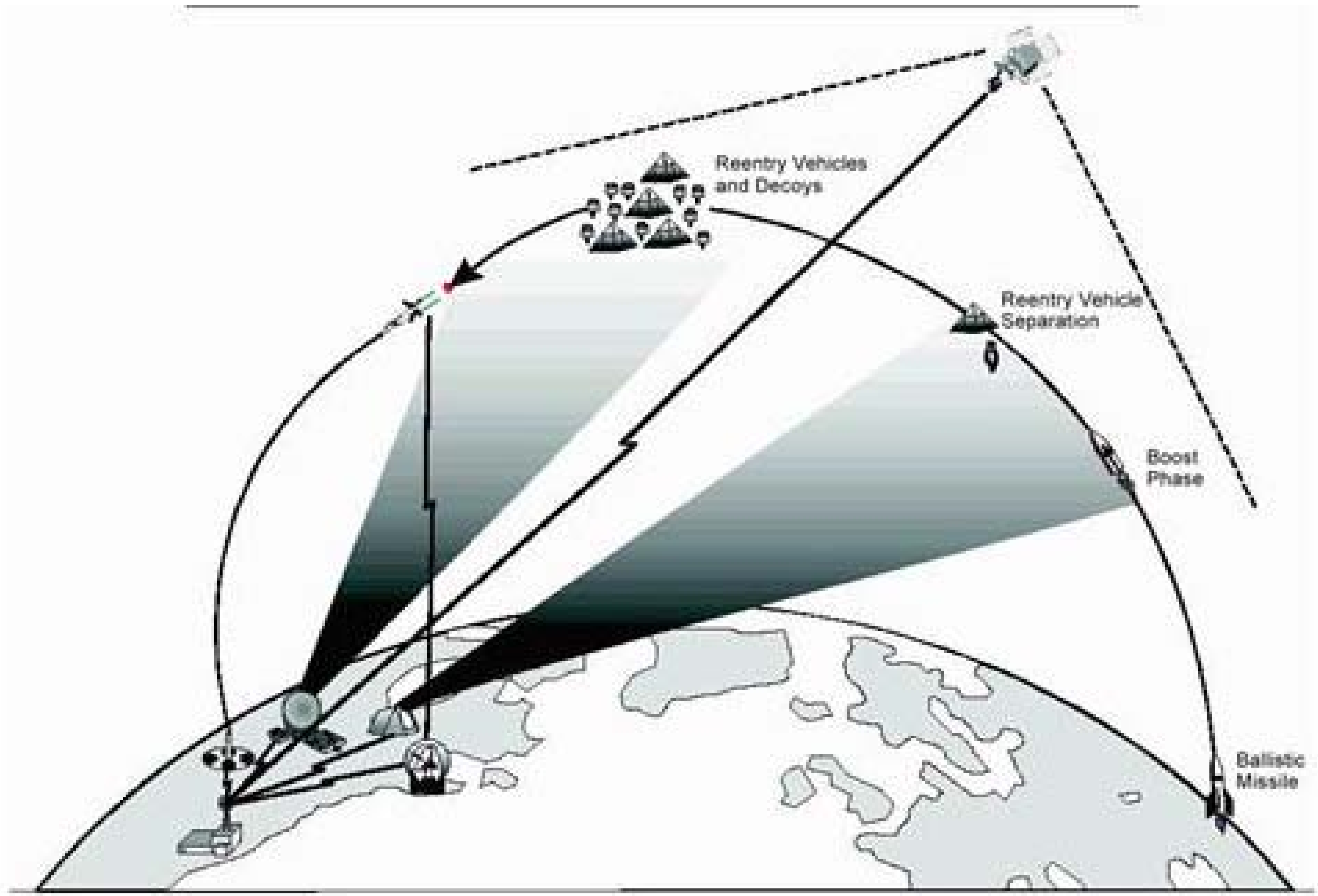


Source of document: Federation of American Scientists website, www.fas.org

Details of the SoS Source Requirements, Showing Relationships and Attributes

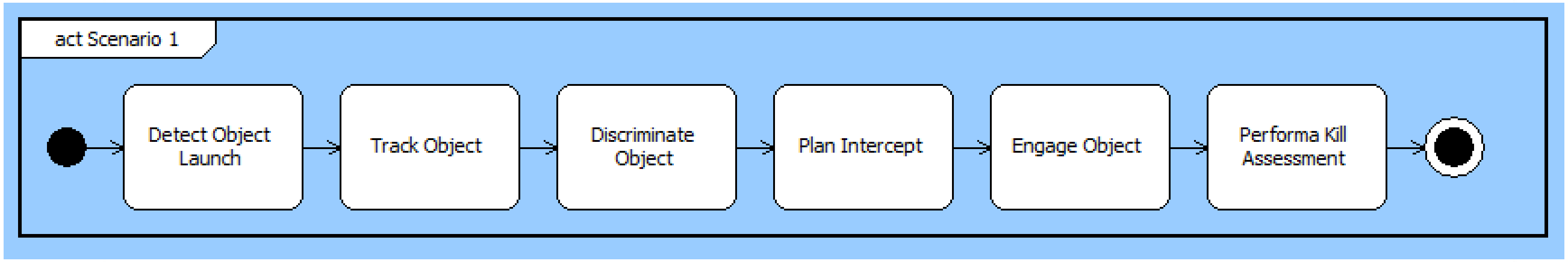
Number & Name	Description	documented by	incorporated in	incorporates
OR.1 Provide ABM defense for the US	Protect the United States against limited ballistic missile threats, including accidental or unauthorized launches or Third World threats. The means to accomplish the NMD mission are: (1) Detect the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

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



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

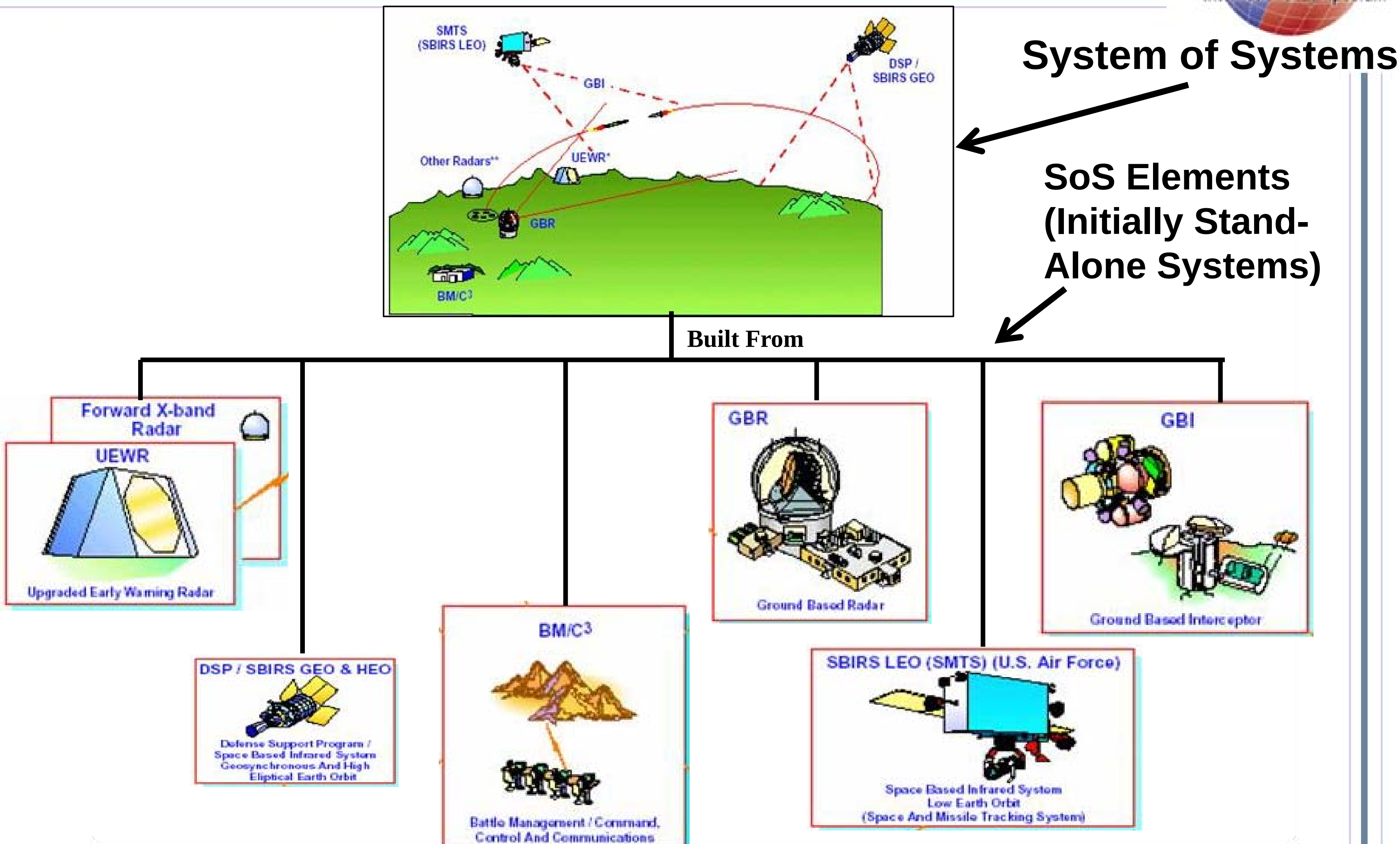
Pre-existing System Elements: What Do You Get?

- 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

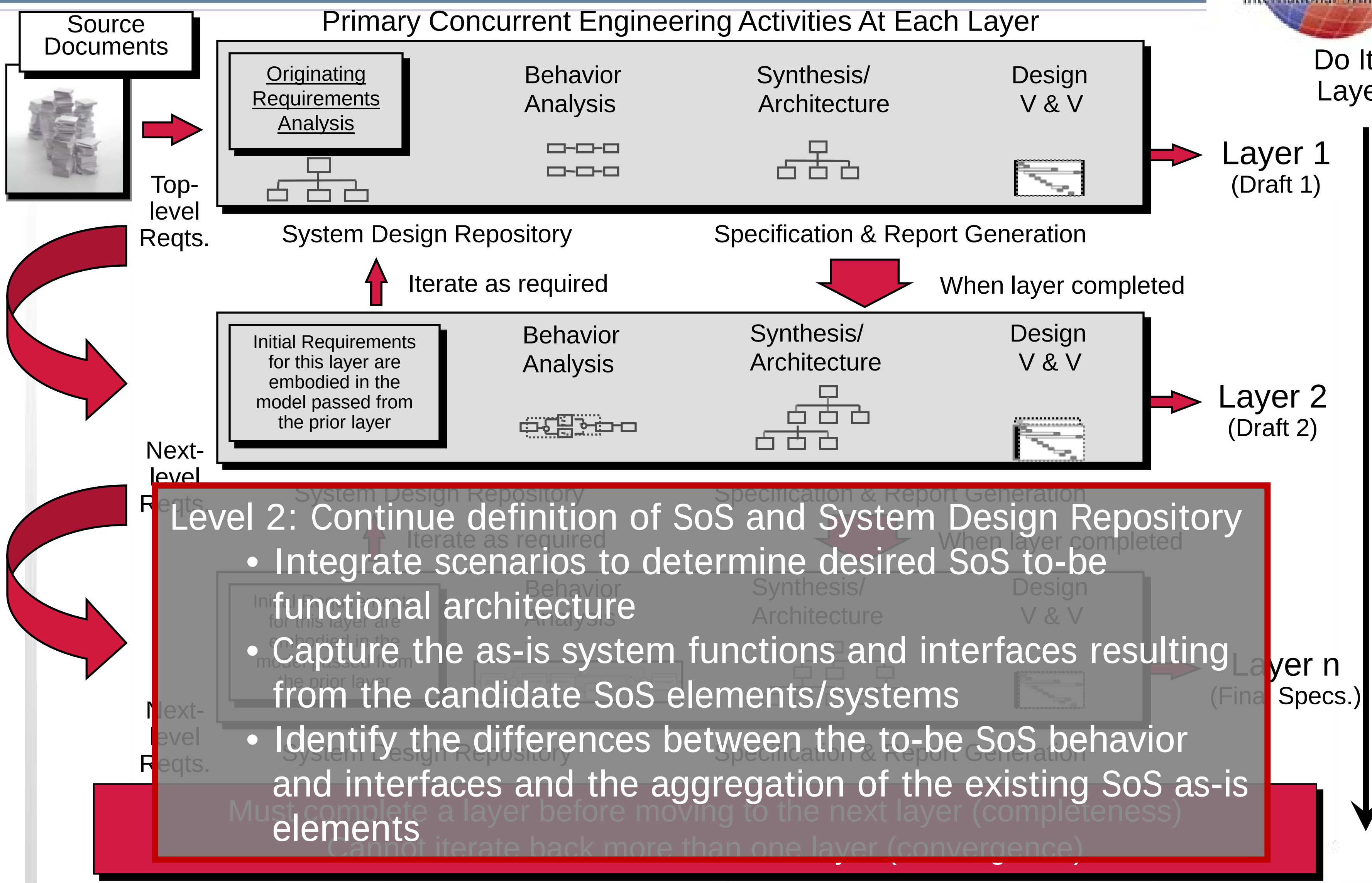
- Cost
- Interfaces
- Stimulus/response
- Agility
  - Command & control
  - Competing resources
  - Error detection and recovery
- Balance of elements: H/W, S/W, & humans
- System evolution

Candidate Elements (Systems) for the Missile Defense System



Source of graphics: Boeing website, www.boeing.com

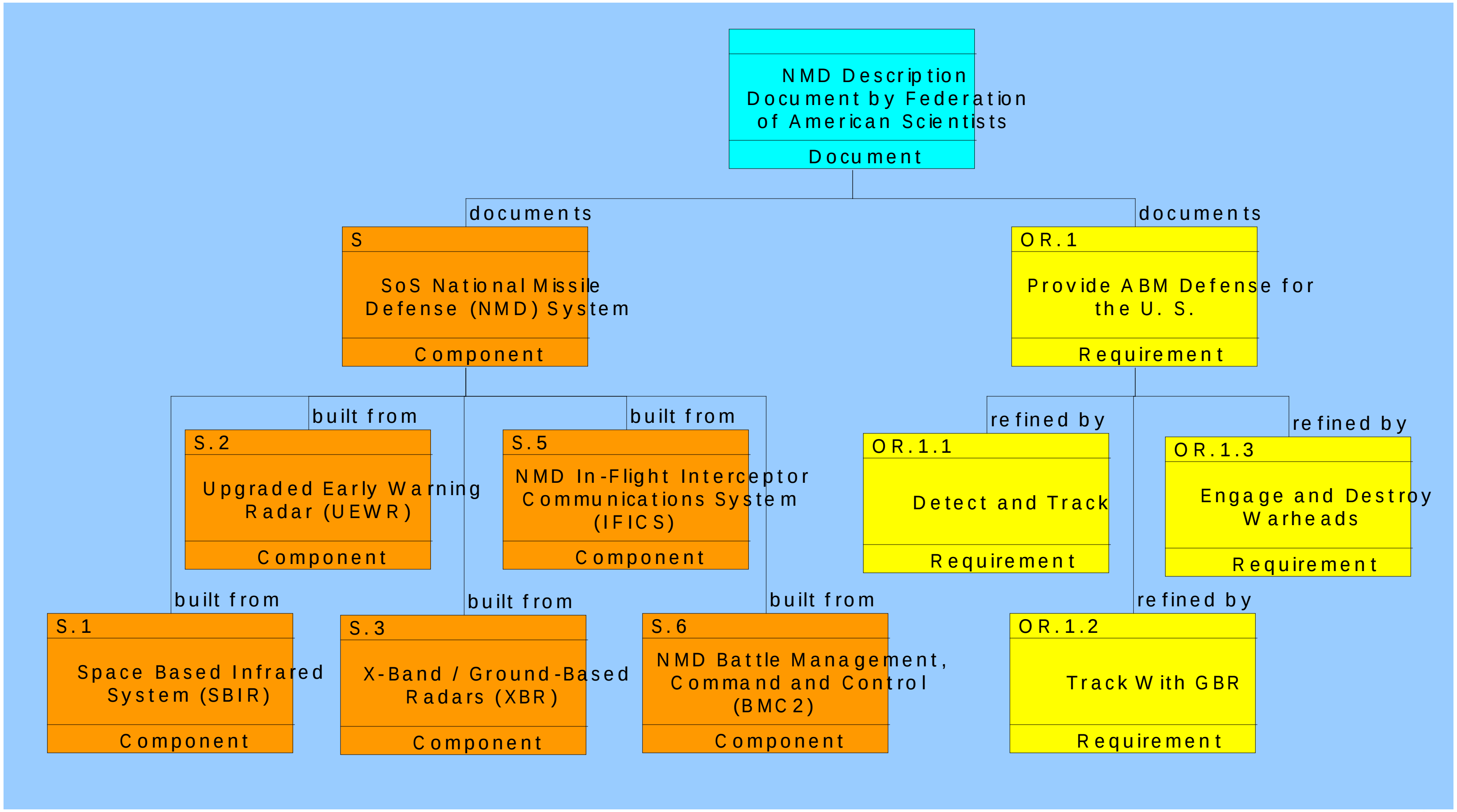
LEVEL 2 of the Onion Model for this SoS Example



Level 2: Continue definition of SoS and System Design Repository

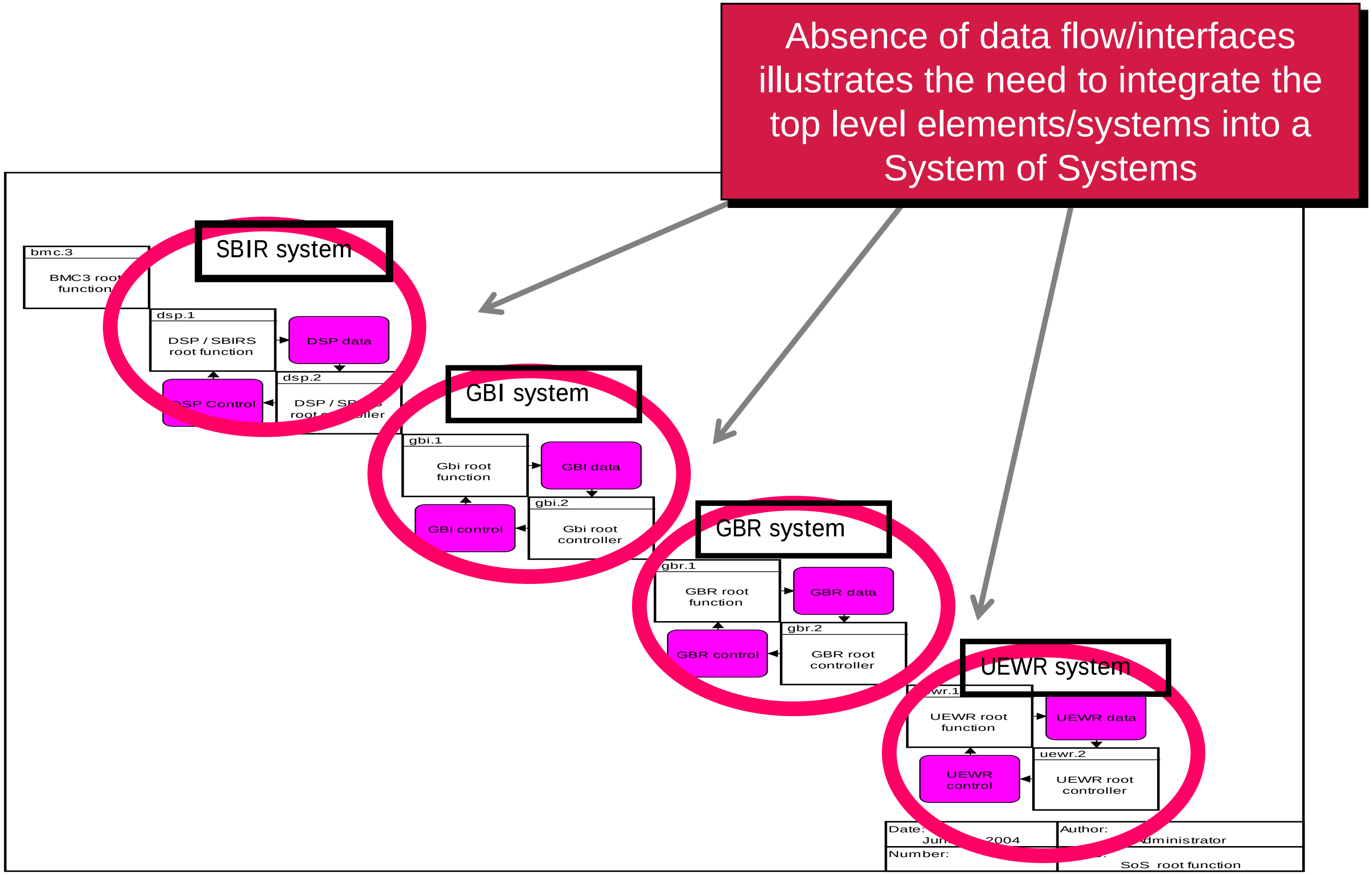
- Integrate scenarios to determine desired SoS to-be functional architecture
- Capture the as-is system functions and interfaces resulting from the candidate SoS elements/systems
- Identify the differences between the to-be SoS behavior and interfaces and the aggregation of the existing SoS as-is elements

Traceability Hierarchy Including the Selected SoS System Elements

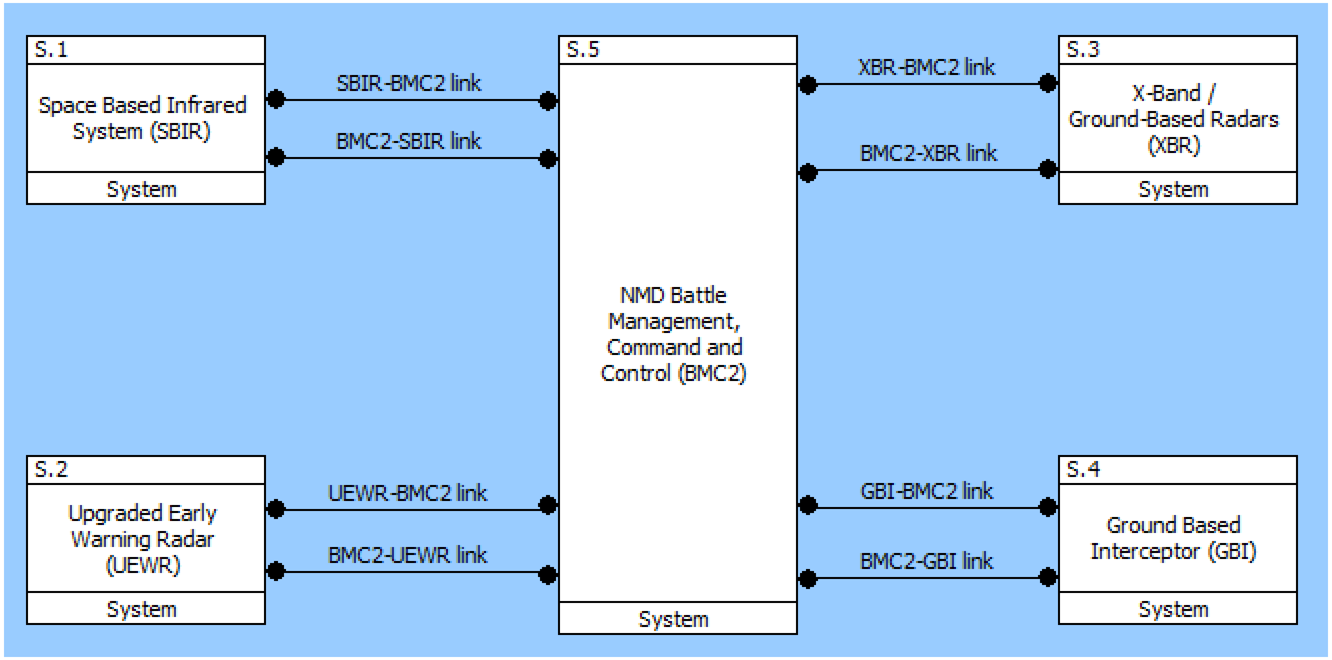


Source of document: Federation of American Scientists website, www.fas.org

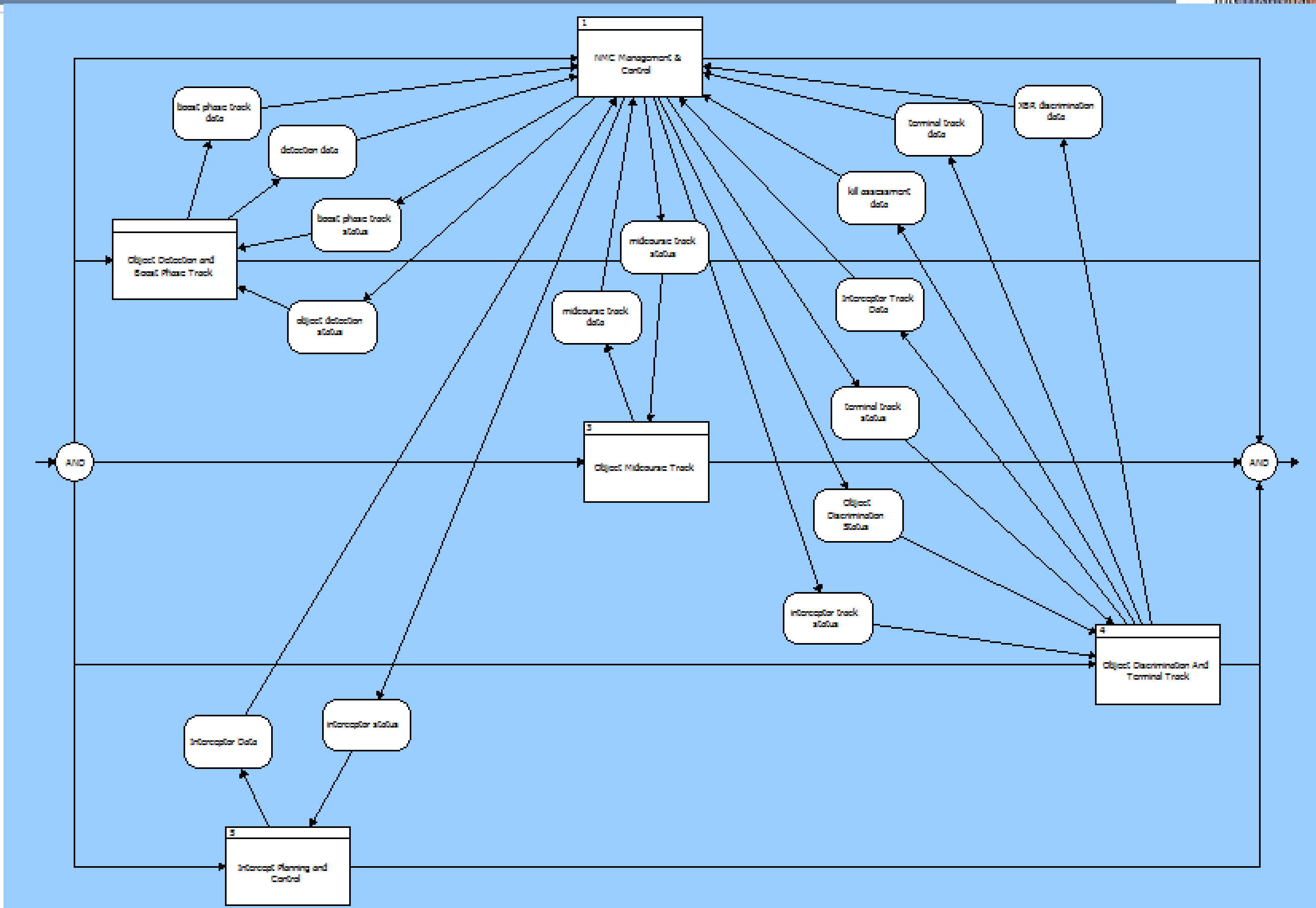
N² Model Highlights Problems with Communication and Coordination Across the SoS System Elements



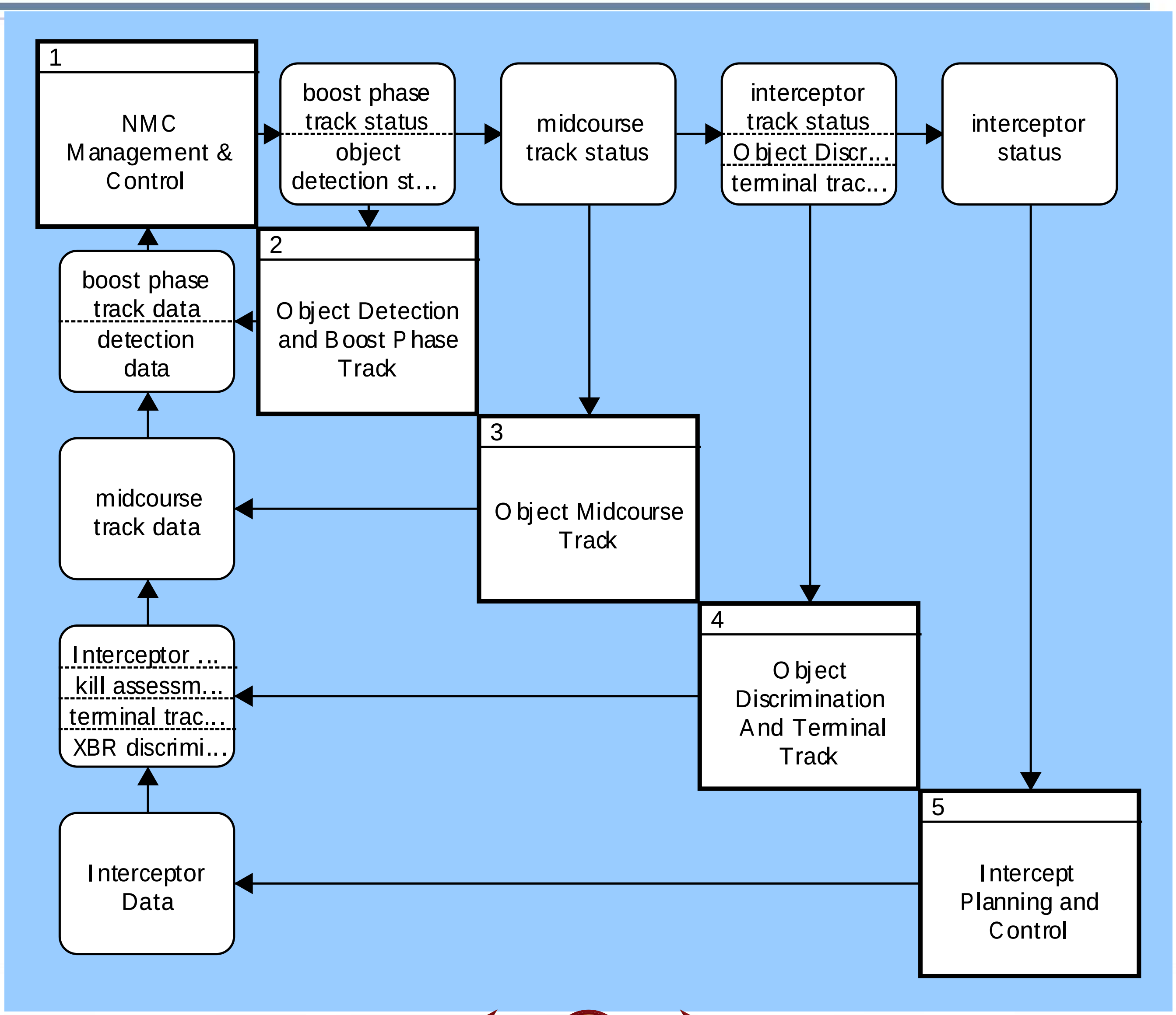
Derived Physical Links are Shown via a Physical Block Diagram



EFFBD Shows SoS Top Level Functional Architecture and Interfaces



N2 Diagram Shows SoS Top-Level Interfaces, Independent of Time



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

Final Thoughts on Applying MBSE to Systems of Systems

- 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

MBSE Benefits to the Enterprise, cont.

- 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

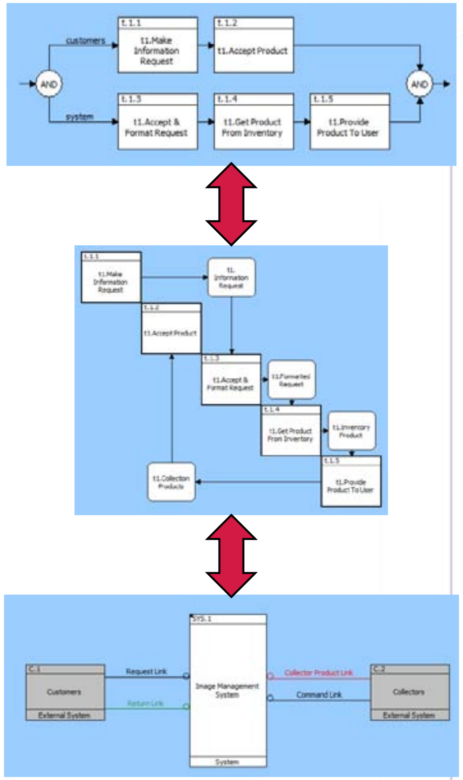
The Efficient SE Process

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 <u>Synchronized</u> 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

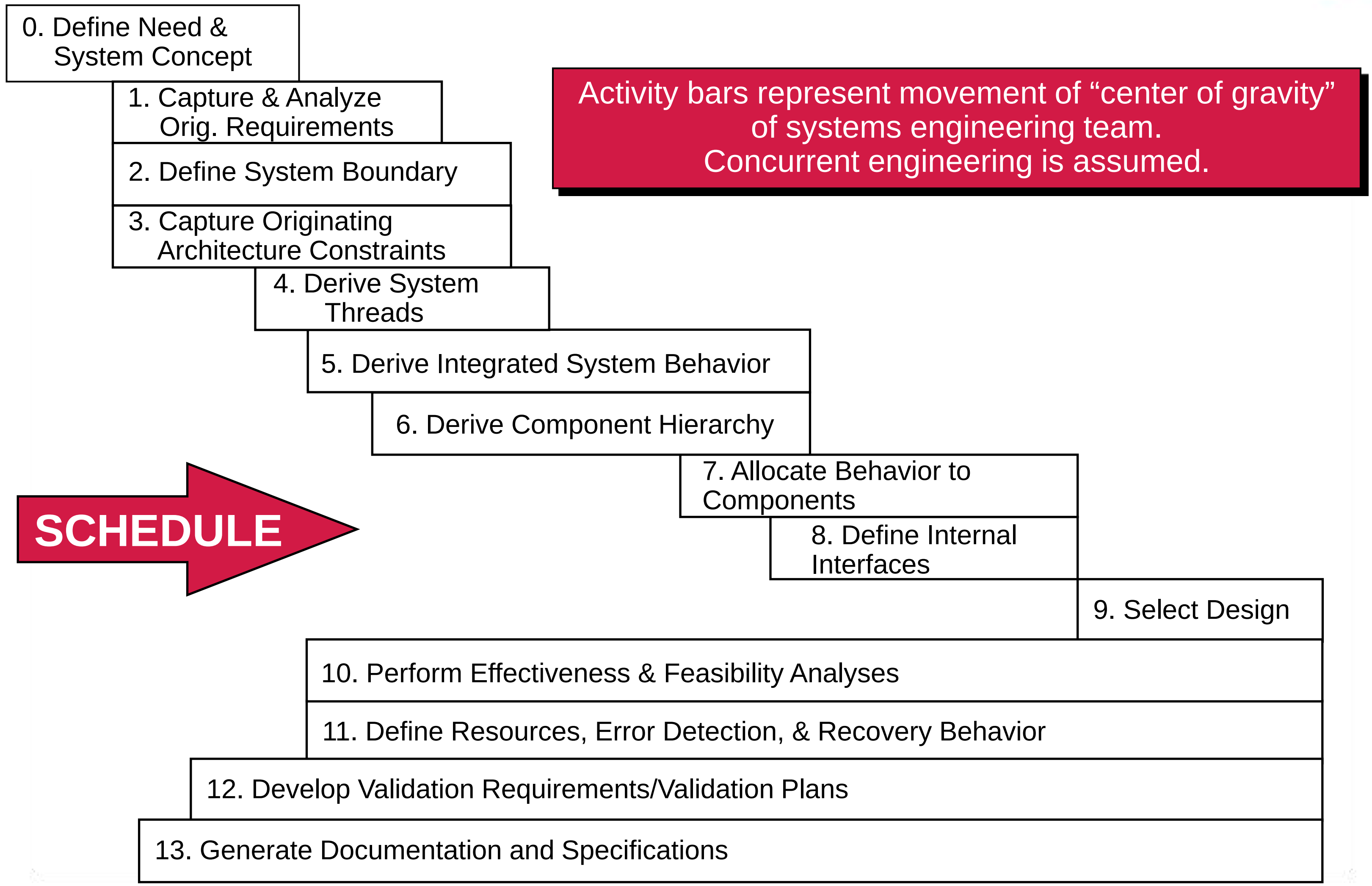


What about performance requirements / resources?

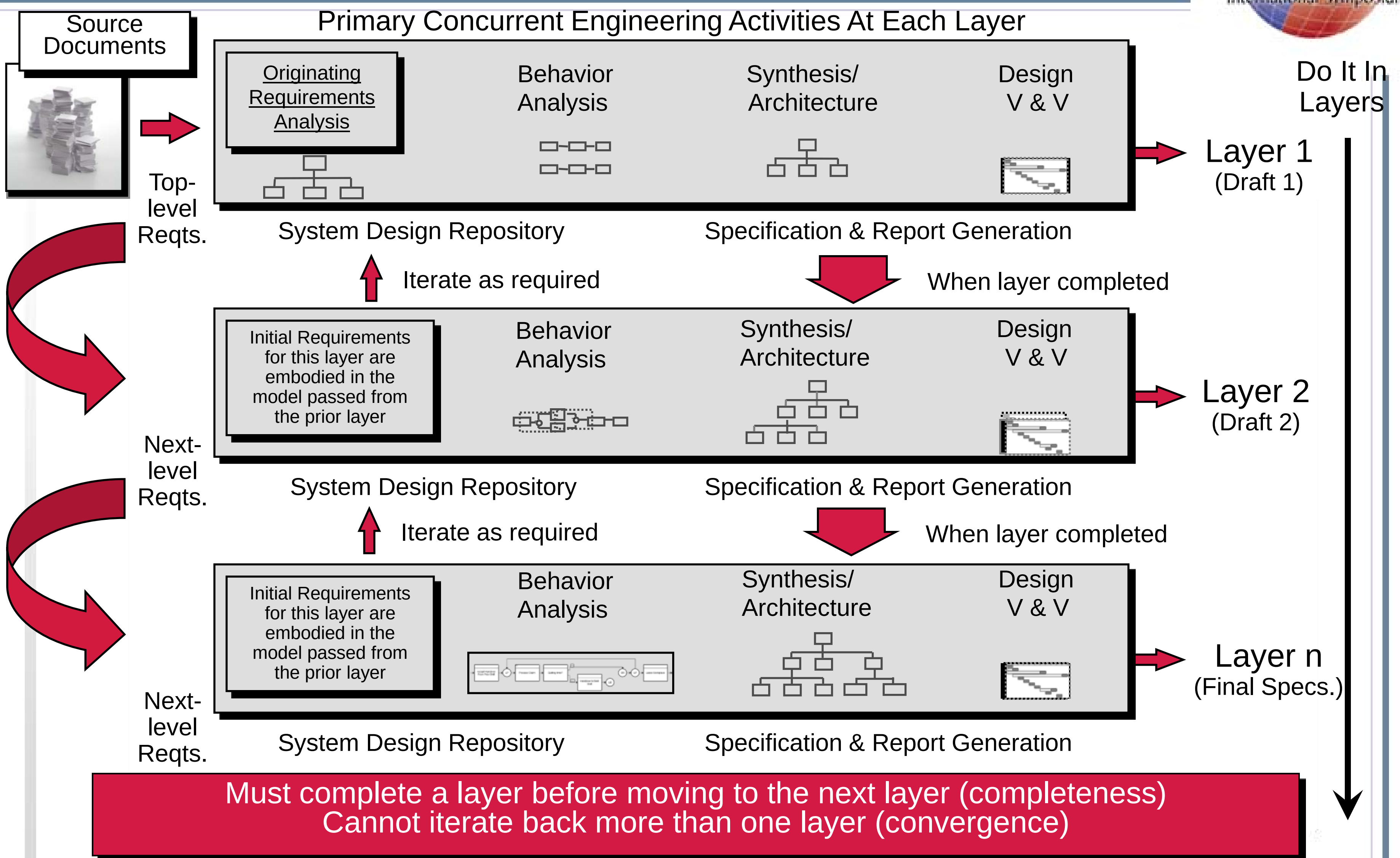
– Captured with parts/combinations of the above models

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



MBSE using the Onion Model. Do Systems Engineering in Increments/Layers



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

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

[1] Executable behavior eliminates structural or dynamic inconsistencies from the requirements.

How Do You Know When You Are Done (at Each Layer)?

Process Elements	Completion Criteria
1. Originating Requirements	1. Agreement on Acceptance Criteria
2. Behavior / Functional Architecture	2. Each function is uniquely allocated to at most one component
3. Physical Architecture Definition	3. Segment/component specs are complete requirements documents
4. Qualification	4. V&V requirements have been traced to test system components

How Do You Know WhenYou 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