NDIA NATIONAL DEFENSE INDUSTRIAL ASSOCIATION

AN EFFECTIVE UML PROCESS

Presented at the
NDIA System Engineering Conference
24 October 2006
By
Jeffrey O. Grady
President JOG System Engineering
6015 Charae Street
San Diego, CA 92122
jgrady@ucsd.edu

Who Is Jeff Grady?

CURRENT POSITION

President, JOG System Engineering, Inc.
System Engineering Assessment, Consulting, and Education Firm

PRIOR EXPERIENCE

1954 - 1964 U.S. Marines

1964 - 1965 General Precision, Librascope Div
- Customer Training Instructor, SUBROC and ASROC ASW Systems

1965 - 1982 Teledyne Ryan Aeronautical
- Field Engineer, AQM-34 Series Special Purpose Aircraft
- Project Engineer, System Engineer, Unmanned Aircraft Systems

1982 - 1984 General Dynamics Convair Division
- System Engineer, Cruise Missile, Advanced Cruise Missile

1984 - 1993 General Dynamics Space Systems Division
- Functional Engineering Manager, Systems Development

FORMAL EDUCATION

BA Math, SDSU
MS Systems Management, USC

INCOSE First Elected Secretary, Founder, Fellow

AUTHOR System Requirements Analysis (1993, 2006), System Integration, System Validation and Verification, System Engineering Planning and Enterprise Identity, System Engineering Deployment

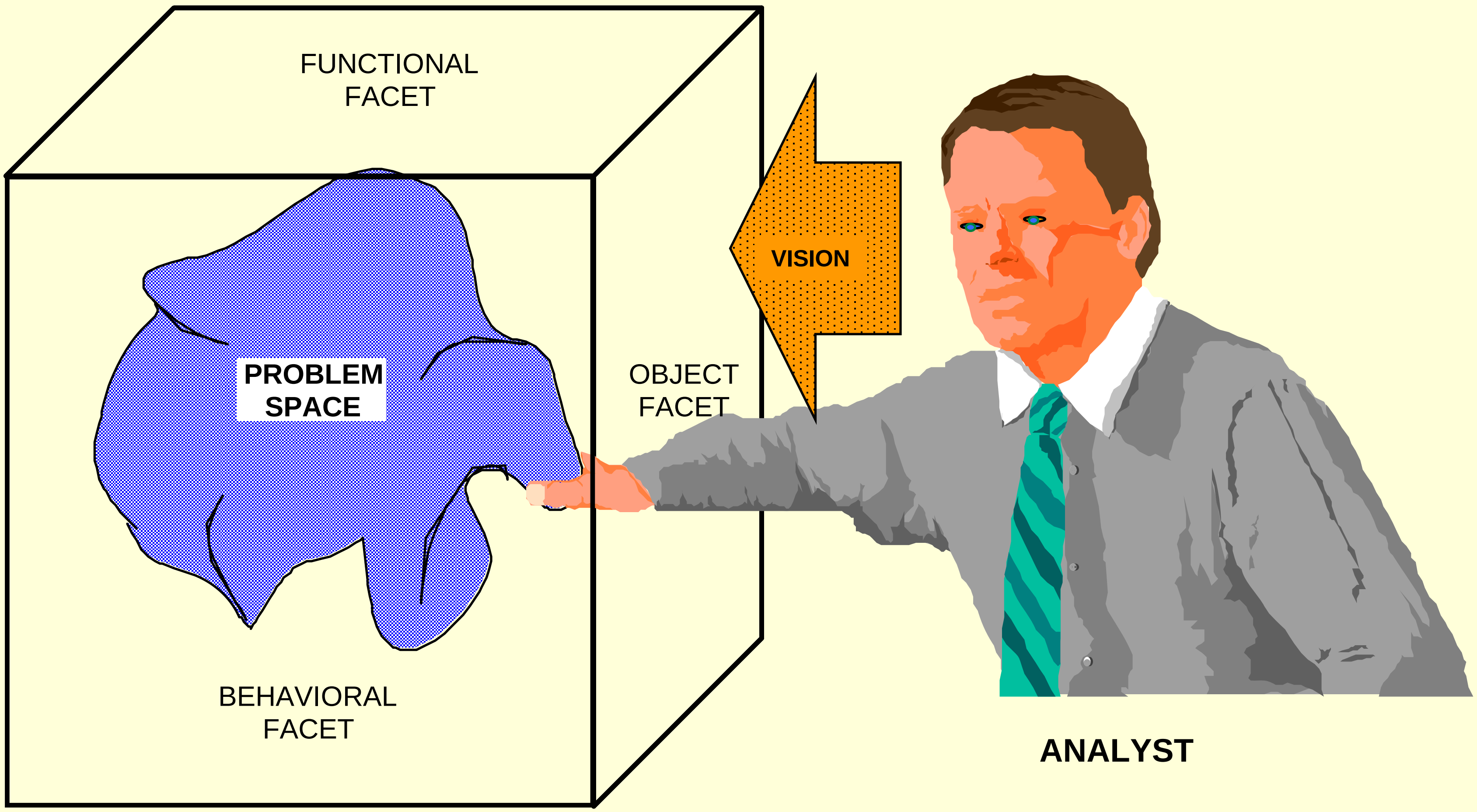
There’s a Problem Out There

- There are problems in the way UML is often applied
- The application of this powerful model should:
  - Follow Sullivan’s encouragement for unprecedented developments
  - Identify SW entities and requirements from the top-down so as to coordinate better with the system engineering and HW work
  - Employ a common product entity structure with the system and HW development
  - Provide for hierarchical traceability across the HW-SW gap
- Available DIDs do not clearly coordinate with application of UML
- Models not always saved or configuration managed
- Traceability problems at the HW-SW gap

Agenda

- Fundamentals of structured analysis
- UML fundamentals using a top-down approach
- Hardware-software requirements traceability
- The future of requirements analysis modeling

Structured View of a Problem Space



Structured Analysis Methods Comparison

MULTI-FACETED APPROACHES	PRODUCT ENTITY FACET	FUNCTIONAL FACET	BEHAVIOR FACET
TRADITIONAL STRUCTURED ANALYSIS	PRODUCT ENTITY BLOCK DIAGRAM	FUNCTIONAL ● FLOW DIAGRAM	SCHEMATIC BLOCK DIAGRAM
MODERN STRUCTURED ANALYSIS	HIERARCHICAL DIAGRAM	DATA FLOW ● DIAGRAM	P SPEC, STATE DIAGRAM
EARLY OBJECT-ORIENTED ANALYSIS	CLASS AND ● OBJECT DIAGRAM	DATA FLOW DIAGRAM	STATE DIAGRAM
UML	CLASS/OBJECT , COMPONENT, & DEPLOYMENT DIAGRAMS	USE CASES AND ACTIVITY ● DIAGRAMS	STATE, SEQUENCE, AND COMMUNICATION DIAGRAMS

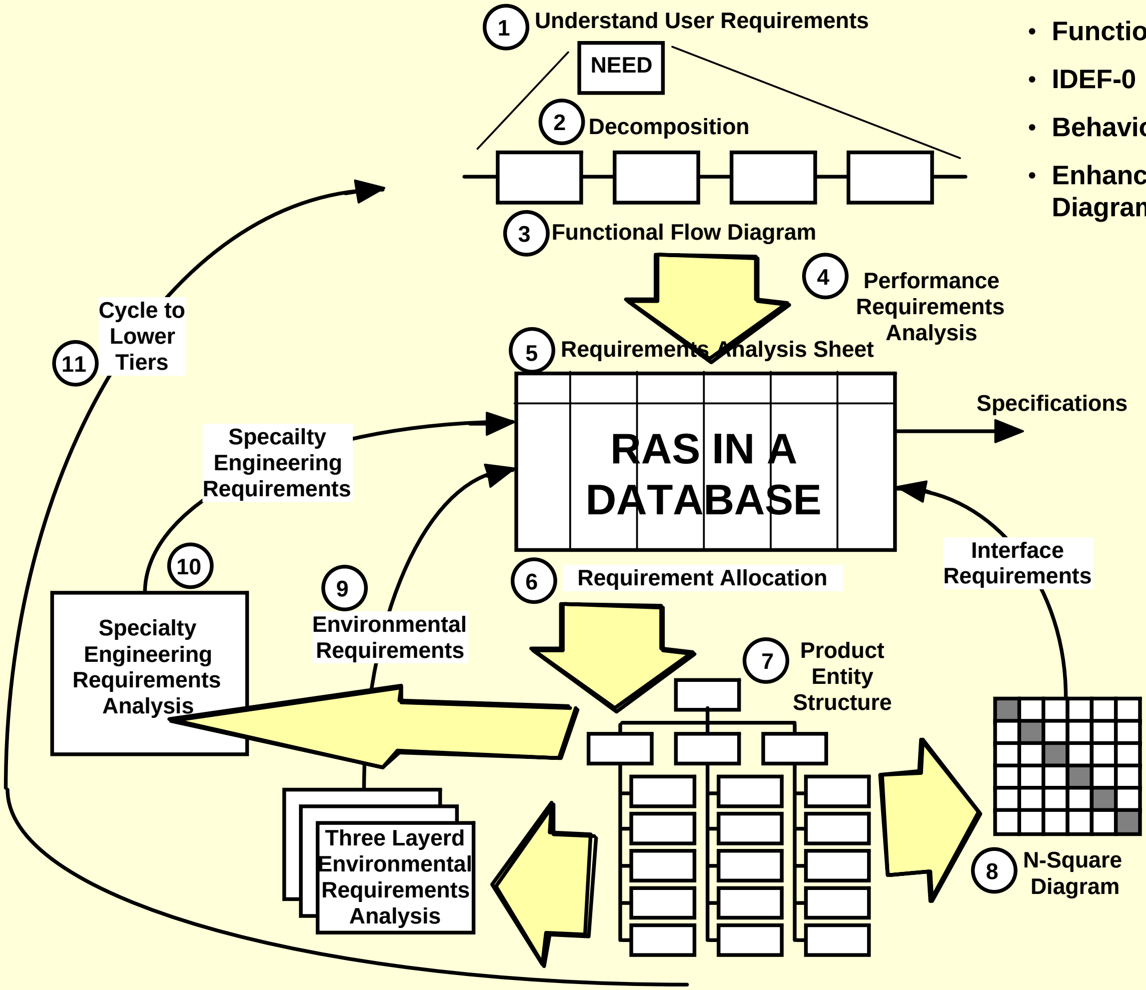
● **UNPRECEDENTED ANALYTICAL ENTRY FACET**

How Should I Enter Problem Space?

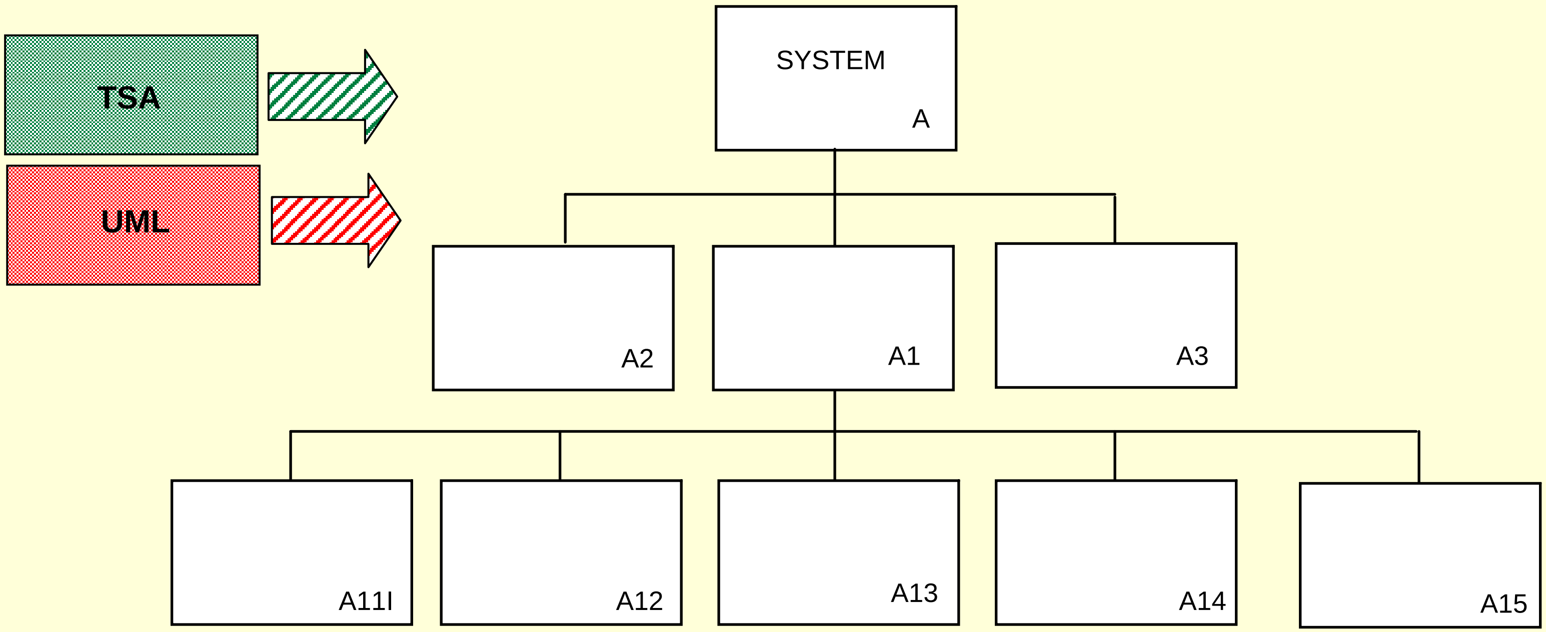
- It is not clear how one can have a system that at the highest level is software
- Software must operate inside of some kind of hardware entity
- Therefore, I would elect to use a system modeling approach for initial problem space entry where the need is the ultimate function
- Traditional structured analysis is such an approach

Traditional Structured Analysis Model Overview

- Functional Flow Diagrams
- IDEF-0
- Behavioral Diagrams (RDD)
- Enhanced Functional Flow Block Diagrams (CORE)



Common Product Entity Structure



- Use a common structure that includes hardware and software.
- Apply a top-down analysis for both that contributes to the identification of entities in the common product entity structure.

A Preferred Modeling Order

- Early object oriented analysis encouraged this pattern.

→ UNDERSTAND CLASSIFIERS FROM THE BOTTOM-UP → DYNAMICALLY MODEL CLASSIFIERS → PACKAGE CLASSIFIERS FROM THE BOTTOM-UP →

- We will follow Sullivan’s encouragement in this case - form follows function - because it coordinates with traditional structured analysis.

→ DYNAMICALLY MODEL THE PROBLEM SPACE FROM THE TOP-DOWN → IDENTIFY RESPONSIBLE CLASSIFIERS → PACKAGE CLASSIFIERS FROM THE TOP-DOWN →

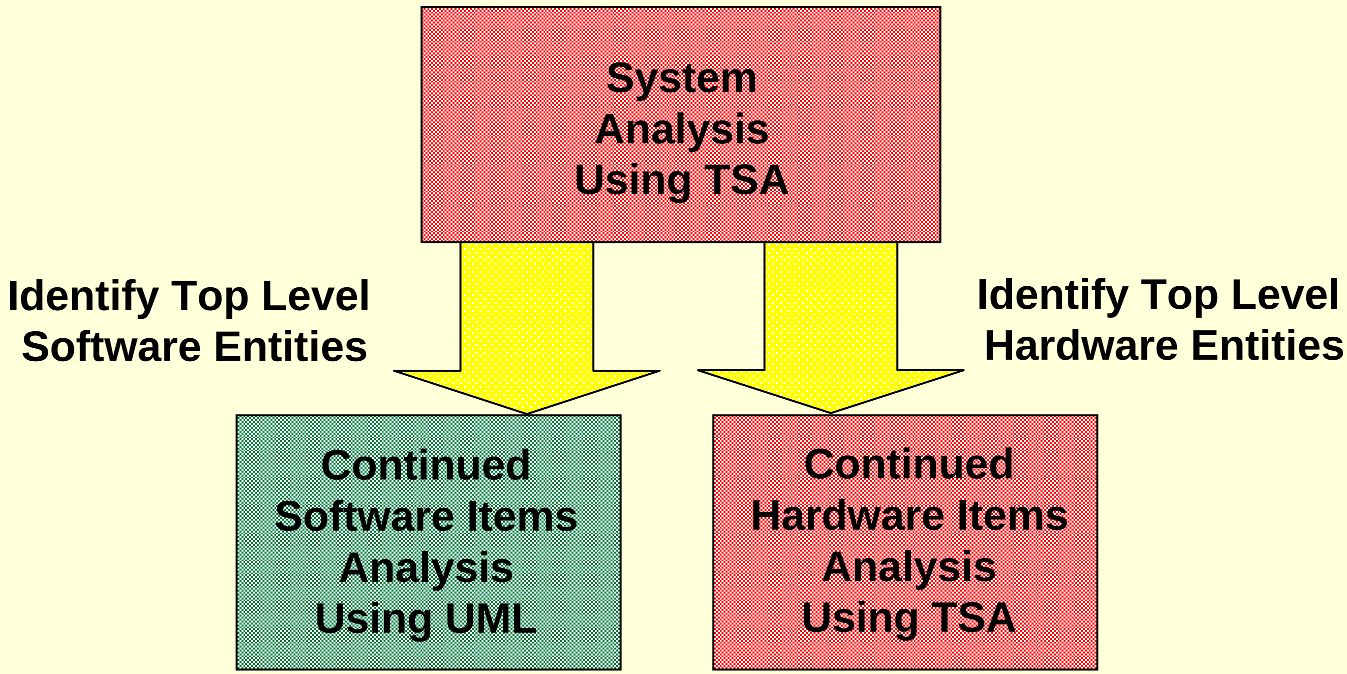
- UML can actually support either direction like any good modeling approach.

Note: A classifier is a general term for a software product entity represented by a node, component, or class in UML but by a block on the product entity diagram like any other entity.

The Software Development Process Using UML

- Identify an initial product entity that will be developed as computer software using traditional structured analysis.
- Dynamically analyze the entity using UML.
  - Use cases
  - Sequence diagram
  - Communication diagram
  - Activity diagram
  - State diagram
- In the sequence, communication, and activity diagramming analysis you will have to identify next lower tier software product entities.
- And the process continues to expand and move deeper translating problem space into solution space.
- At the bottom are classes about which code can be written based on requirements derived from the dynamic modeling work.

Surprise!



Note that the SW work pattern encouraged exactly parallels that employed in TSA.

The Diagrams of UML 2

- **For modeling dynamic aspects of the system**
  - Use case diagram
  - Sequence diagram
  - Communication diagram (renamed in 2)
  - State diagram
  - Activity diagram
  - Timing diagram
  - Interaction overview diagram (2)
- **For modeling static aspects of the system**
  - Object and class diagrams
  - Component diagram
  - Deployment diagram
  - Composite structure diagram (2)
  - Package diagram (2)

Classifiers

Covered in this discussion

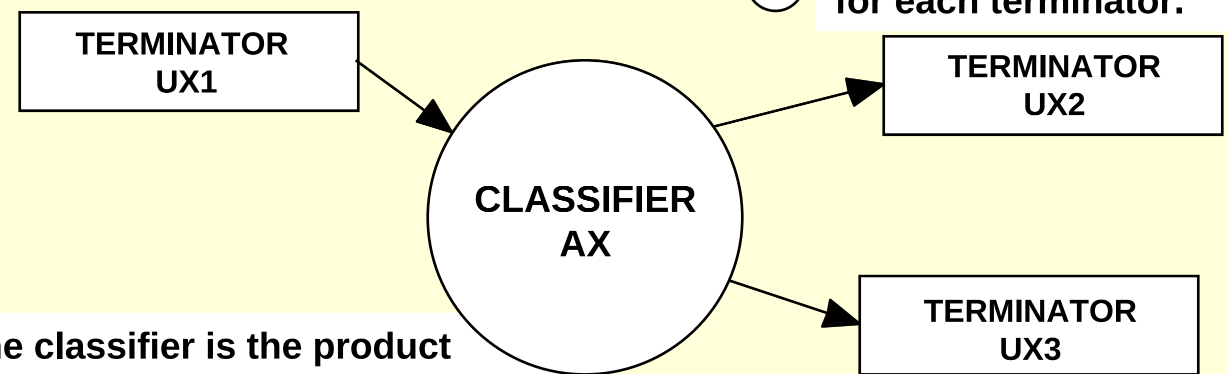
(2) = added in UML 2.0

Context Diagram

Borrowed from Modern Structured Analysis to provide an organized approach to use case identification.

1

1a Identify one or more use cases for each terminator.



1b The terminators reflect necessary external influences between the system and its environment.

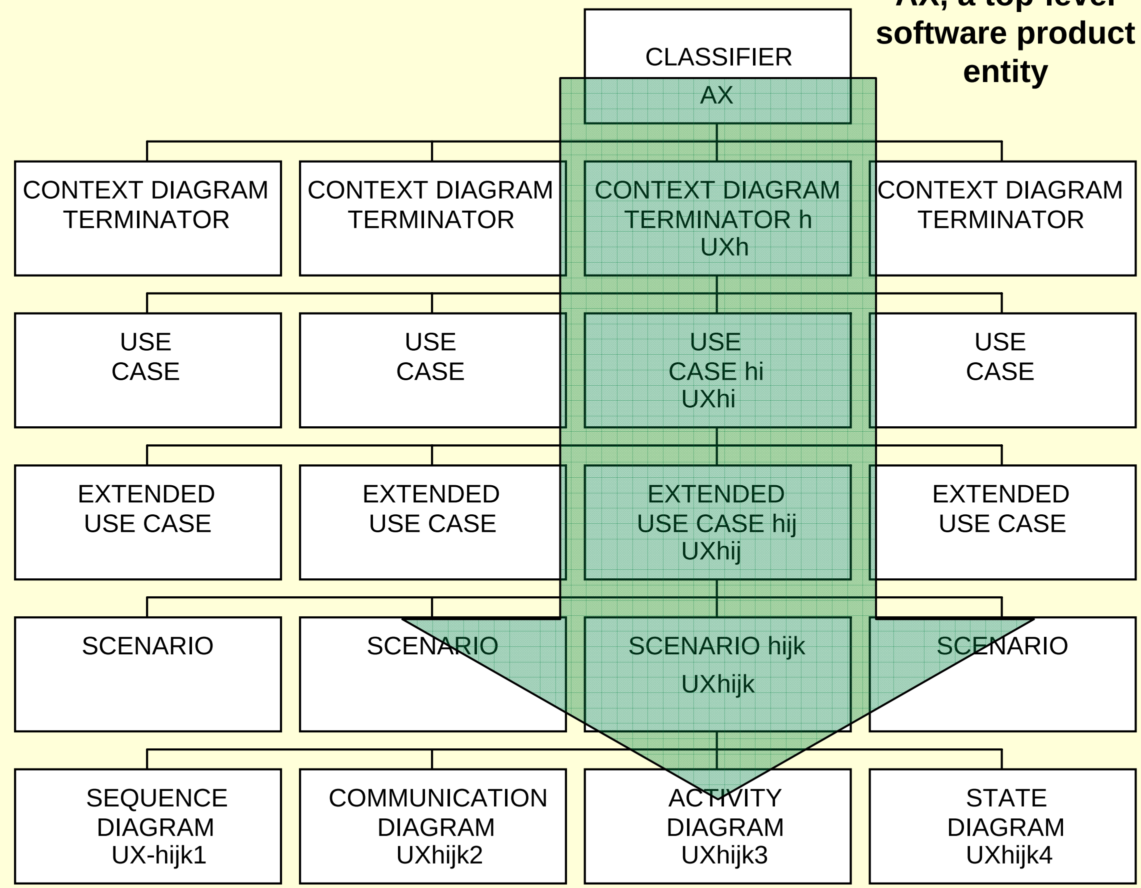
1c The classifier is the product entity the specification is being written for.

Hierarchical Modeling Relationships

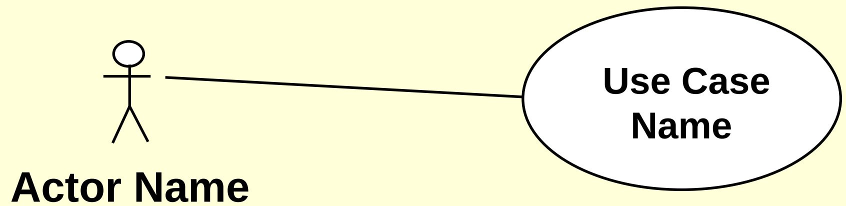
- The Supporting Dynamic Modeling Artifacts

AX, a top-level software product entity

NOTE: Only a single analytical string has been expanded here.

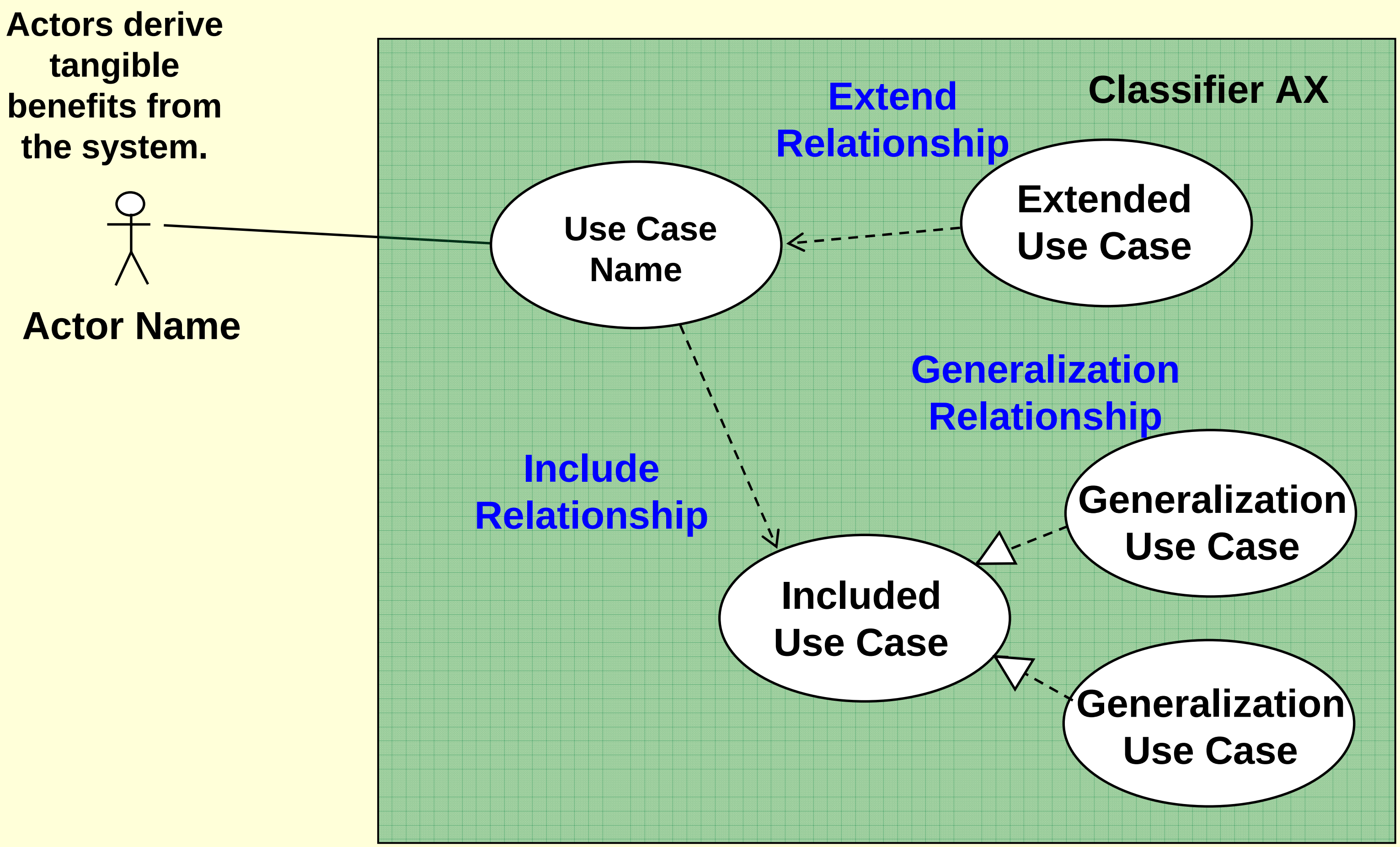


Use Case Fundamentals



- A use case is a more expressive form of the context diagram used in modern structured analysis.
- A use case bubble represents some aspect of the system being developed.
- An actor represents some external agent gaining benefit from the system.

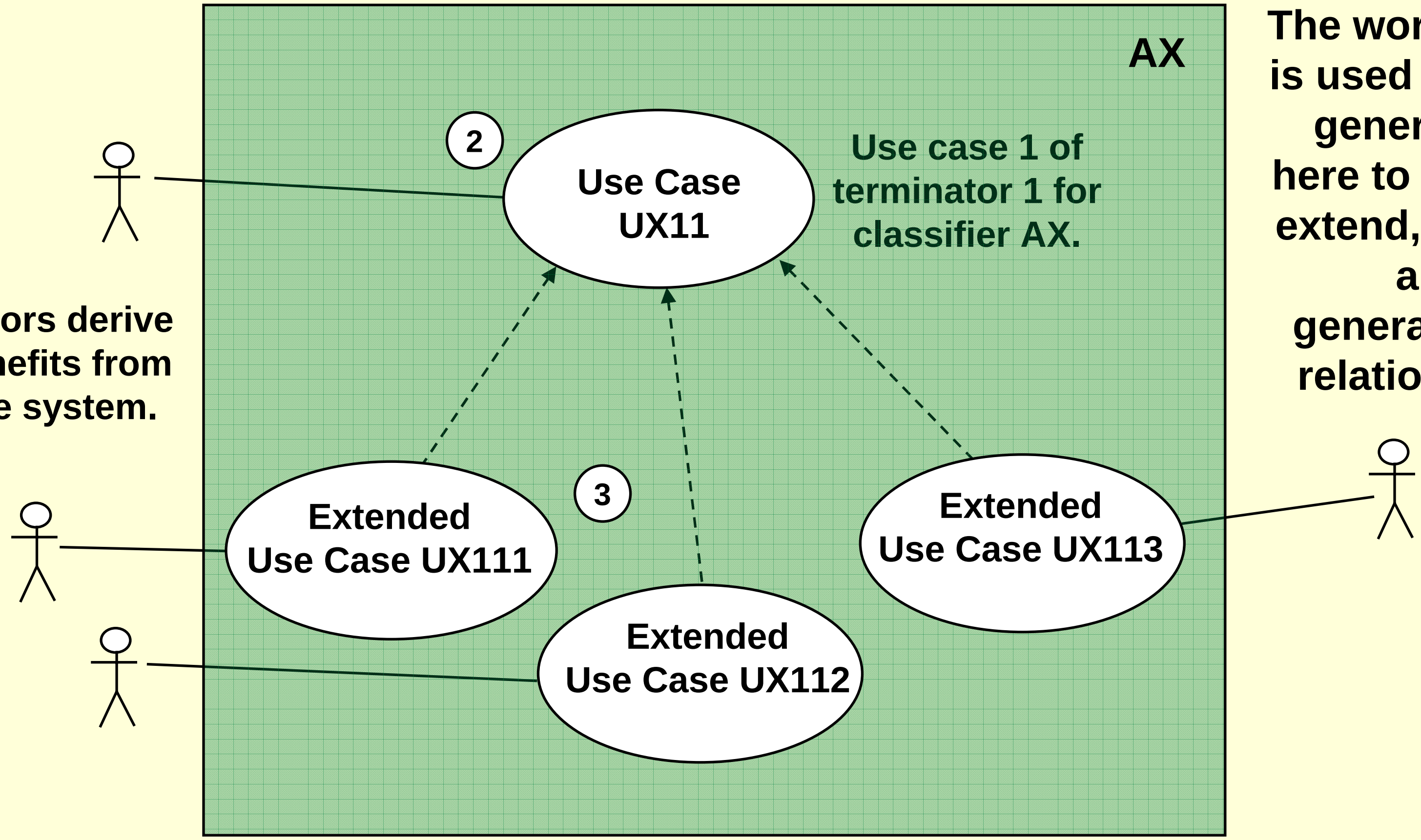
Use Case Relationships



Use Case Relationships

- **Extend**
  - Pushes common behavior into other use cases that extent a base use case
- **Include**
  - Pulls common behavior from other use cases that a base use case includes
- **Generalization**
  - A child use case inherits behavior and meaning of the base use case
  - The child use case may add or override the behavior of the base use case

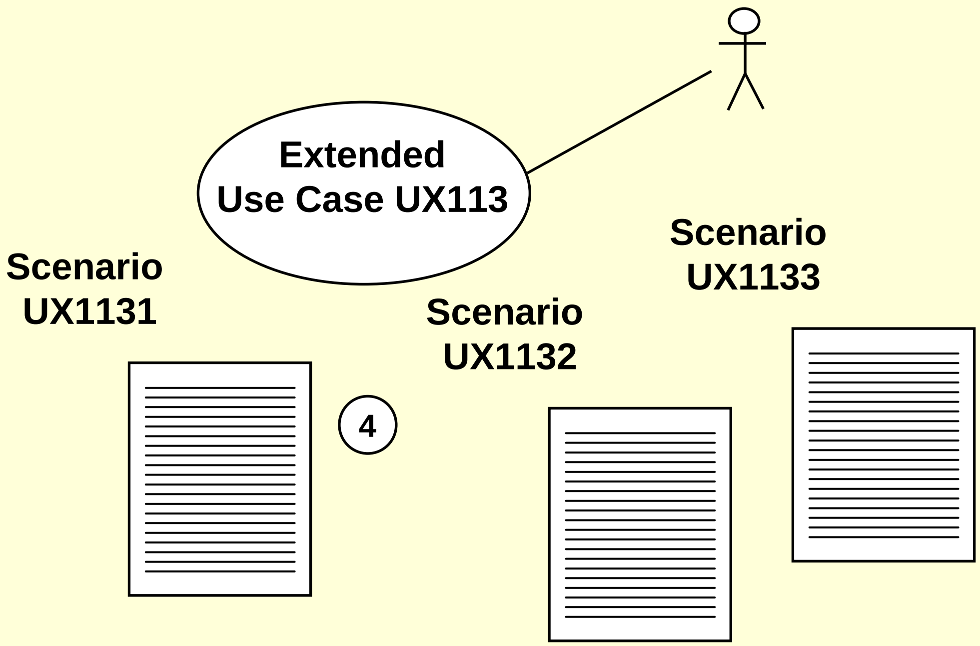
Use Case UX11



Actors derive benefits from the system.

The word extend is used here in a generic way here to embrace extend, include, and generalization relationships.

Possible Multiple Scenarios



The word extend is used in a generic way here to embrace extend, include, and generalization relationships.

Textual scenario descriptions (use case specifications)

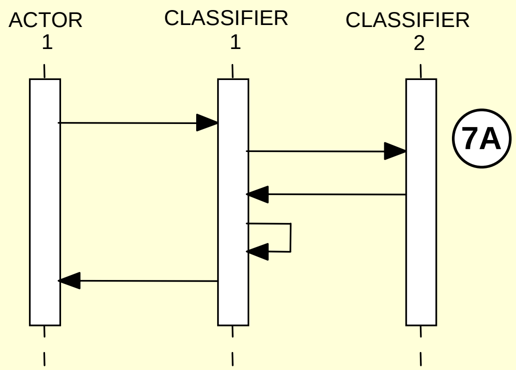
Scenario

- A sequence of actions that illustrates behavior.
- A scenario may be used to illustrate an interaction or execution of a use case instance.
- Text description that can be captured in paragraph 3.1.2.h.i.j.k of the classifier specification.

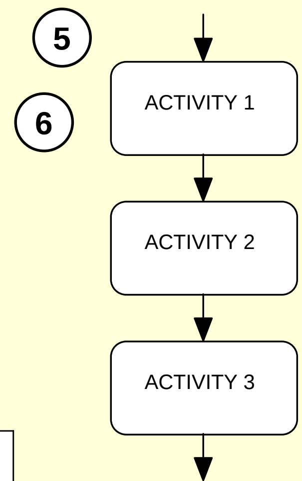
If you want a copy of a DID email jgrady@ucsd.edu

The Other Dynamic Models

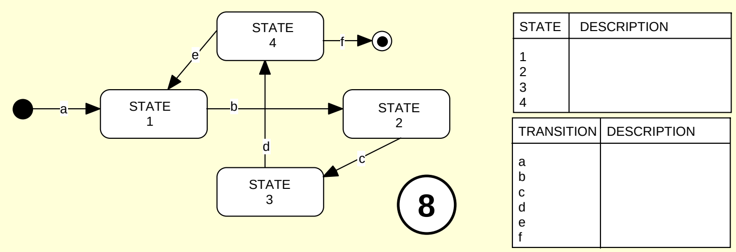
Sequence Diagram UX11321



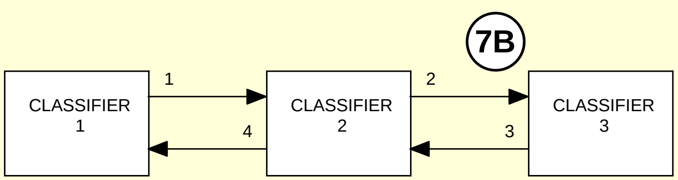
Activity Diagram UX11323



State Diagram UX11324



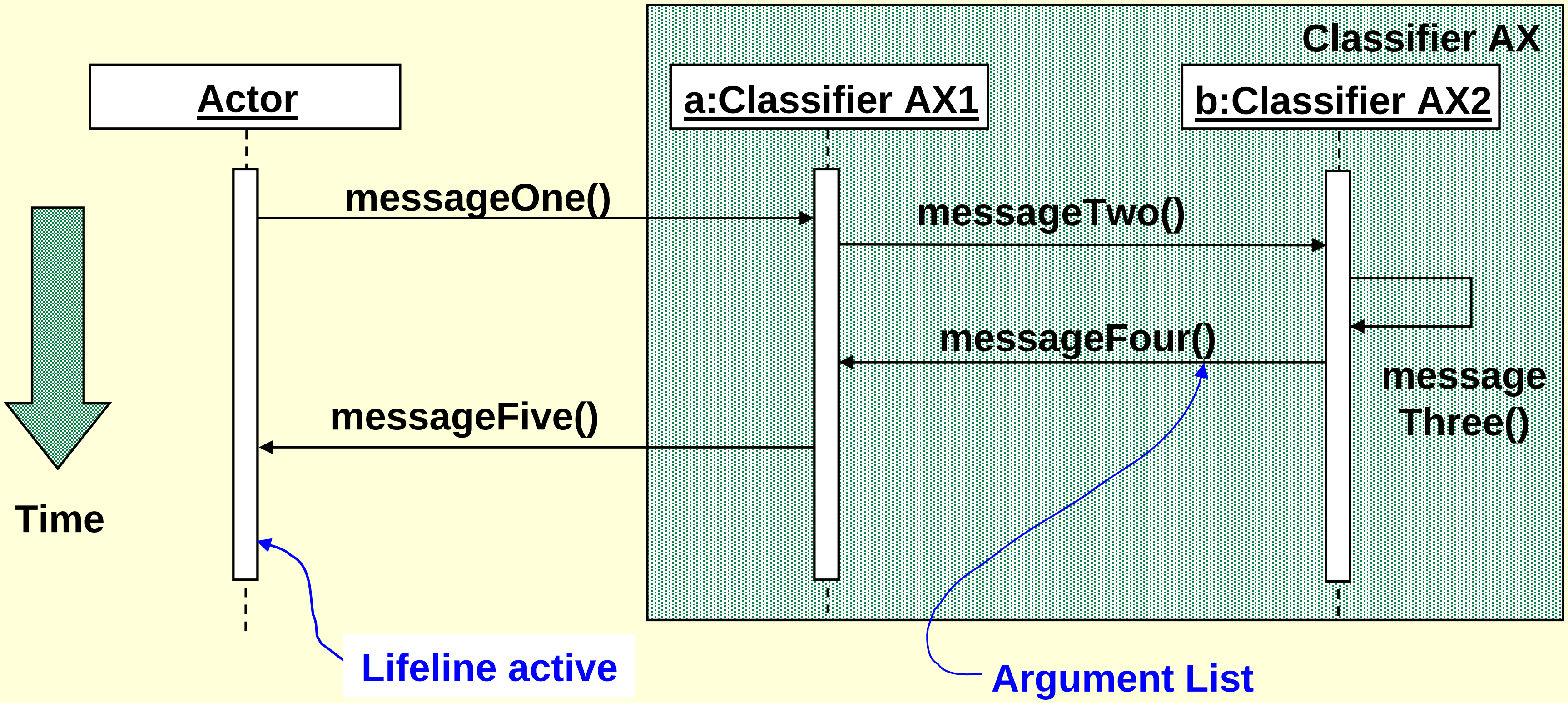
Interaction Diagrams



Communication Diagram UX11322

Sequence Diagram UX11321

Emphasizes the time ordering of messages



It is understood that the classifiers are performing operations, possibly modeled in activity or state diagrams, relative to the message content.

NDIA NATIONAL DEFENSE INDUSTRIAL ASSOCIATION

Messages Between Lifelines

- **A message is the specification of a communication among classifiers on a class or object diagram or between the classifier represented by life lines on the sequence diagram or blocks of a communication diagram.**
- **When a message is passed from one classifier to another some action usually results on its receipt.**
- **The action may result in a change of state in the classifier on the arrow head.**
- **Describe the related requirements in terms related to the target classifier.**

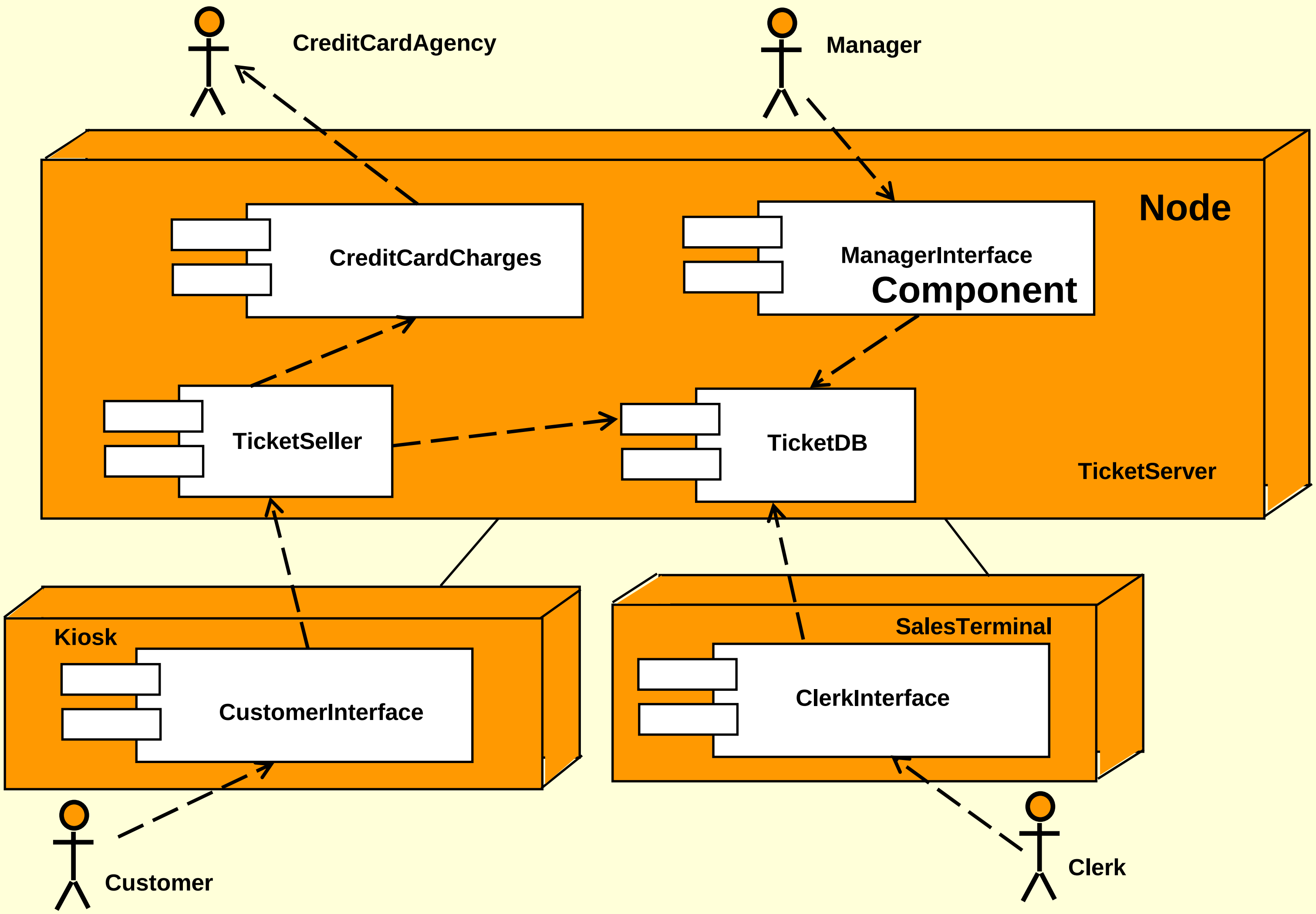
Sequence Diagram Message Types

- **Call**
  - Invokes an operation on an object represented by the lifeline
  - An object can send a call to itself resulting in a local invocation
- **Return**
  - Returns a value to the caller
- **Send**
  - Sends a signal to an object
- **Create**
  - Creates an object
- **Destroy**
  - Destroys an object

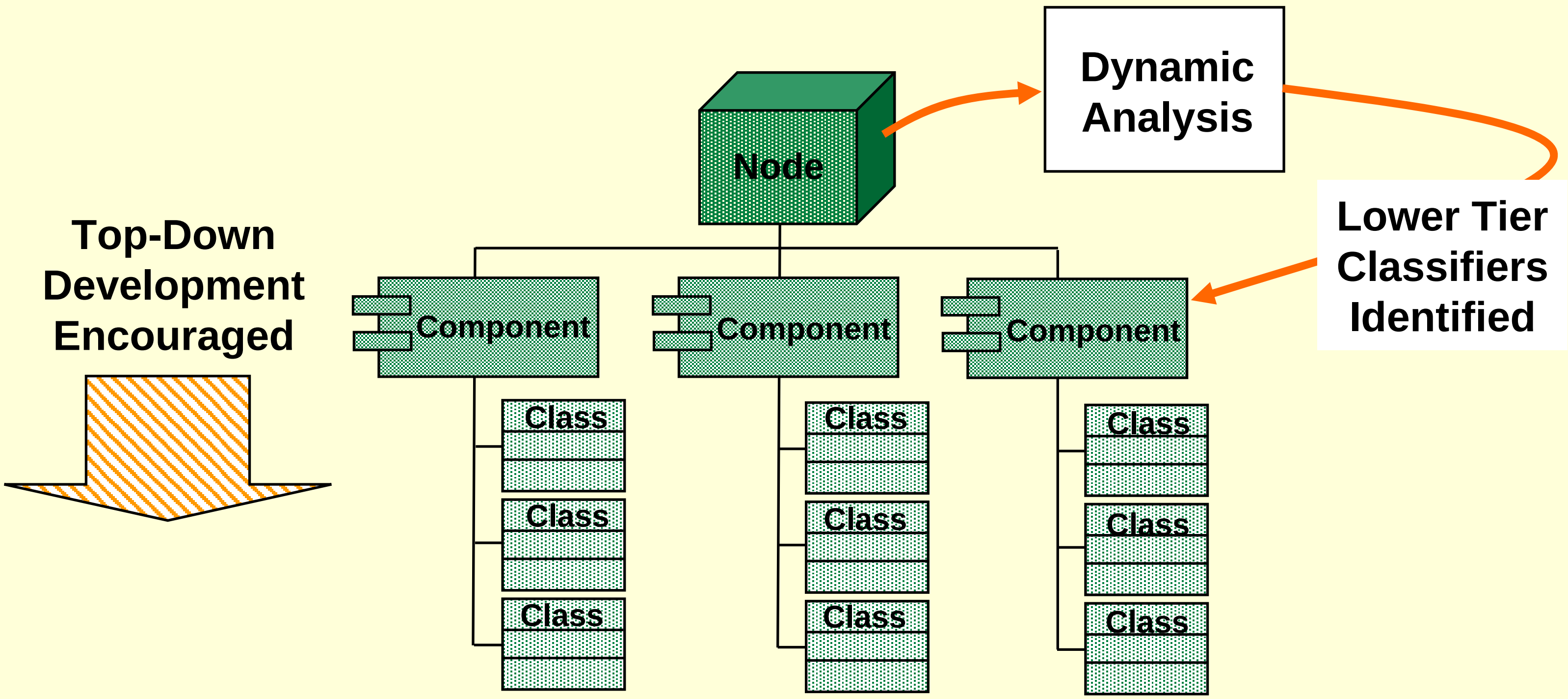
The UML Static Entities

- **System/Subsystem**
  - The highest level software entity. There can be many of these entities in a real system composed of hardware and distributed software. A collection of subsystems composed of nodes or simply nodes.
- **Node**
  - Appears on a deployment diagram that exists at run time and is a computational resource, generally having at least some memory and often processing capability. A collection of components.
- **Component**
  - A modular part of the system consisting of other components or directly of classes.
- **Class**
  - A description of a set of objects that share the same attributes, operations, relationships, and semantics.
- **Object**
  - An instance of a class.

Deployment and Component Diagrams

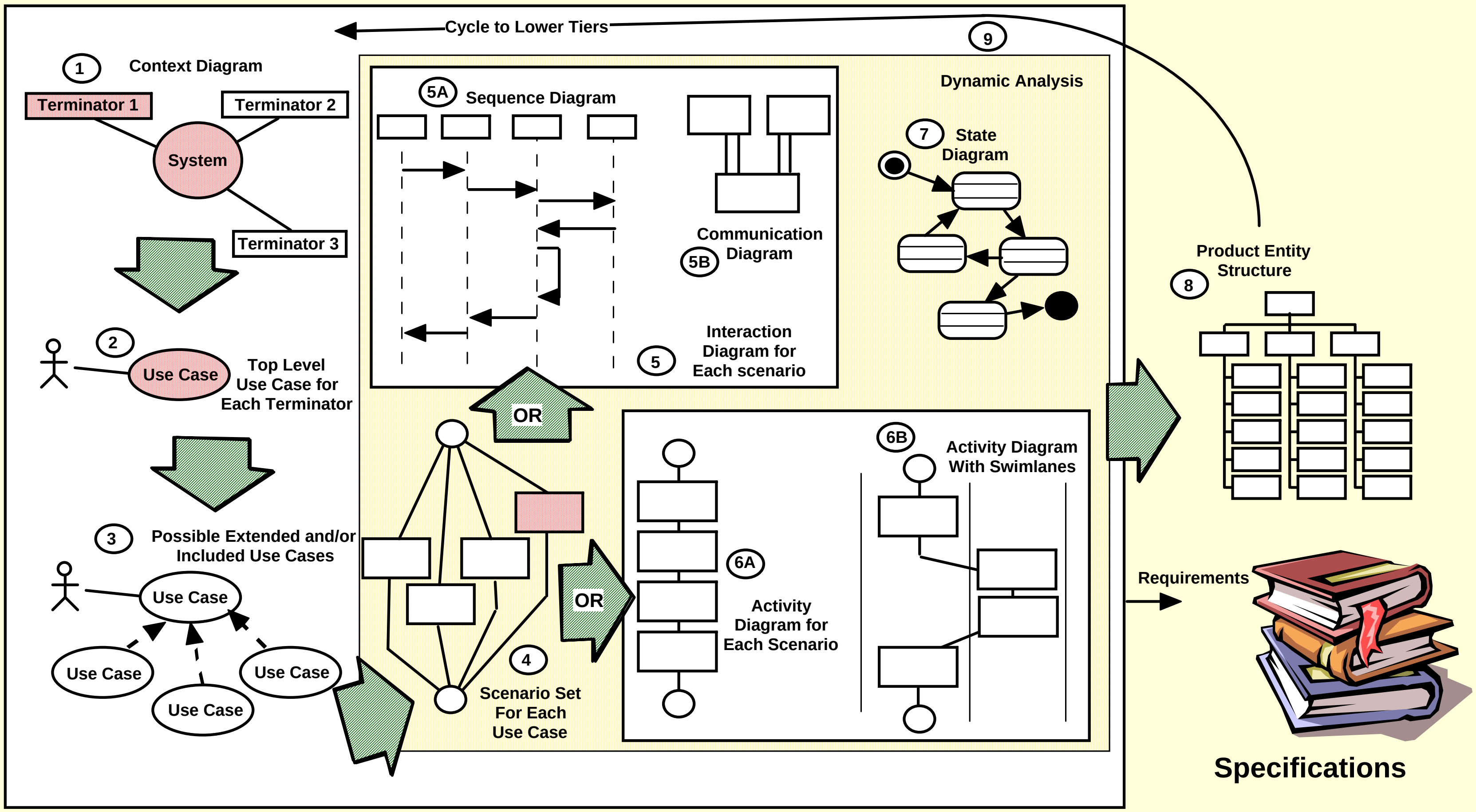


UML Structural Artifacts in a Product Entity Structure



NDIA NATIONAL DEFENSE INDUSTRIAL ASSOCIATION

A Flexible Dynamic Modeling Overview

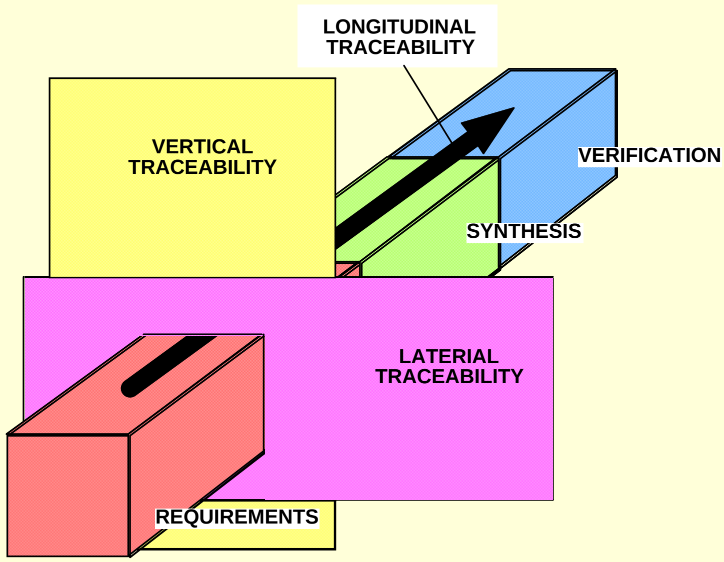


Organizing the Dynamic Modeling

- Use a context diagram to organize the use cases.
- Recognize a family of use cases if necessary.
- If use cases complex, recognize two or more scenarios for each lowest tier use case.
- For each scenario, build a sequence or activity diagram and in the process identify next lower tier classifiers and messages between the actors and lower tier classifiers.
- Apply communication, activity or sequence, and state diagrams as needed.
- Derive requirements from dynamic modeling artifacts and relationships.

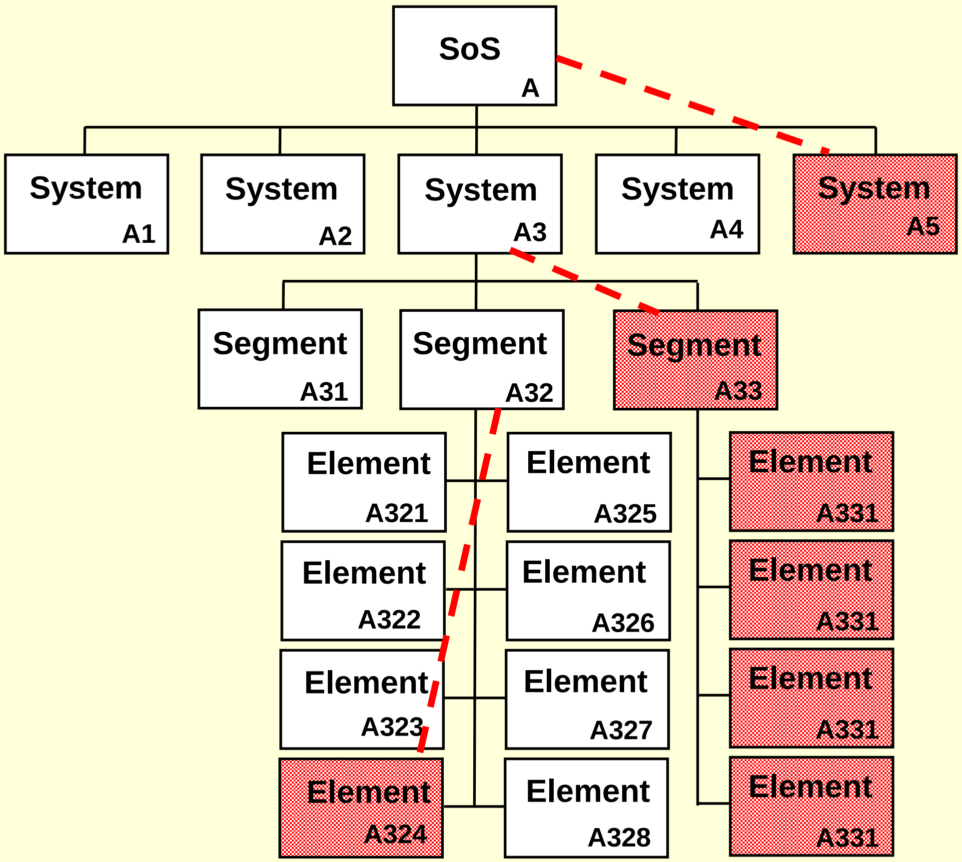
Traceability Forms

- Vertical requirements traceability
  - Hierarchical or parent-child
  - Requirements source traceability
  - Requirements rationale traceability
- Longitudinal traceability
  - Requirements to synthesis and verification
- Lateral traceability
  - Traceability to method
- Applicable document
  - Internal integrity



NDIA NATIONAL DEFENSE INDUSTRIAL ASSOCIATION

System Product Entity Structure



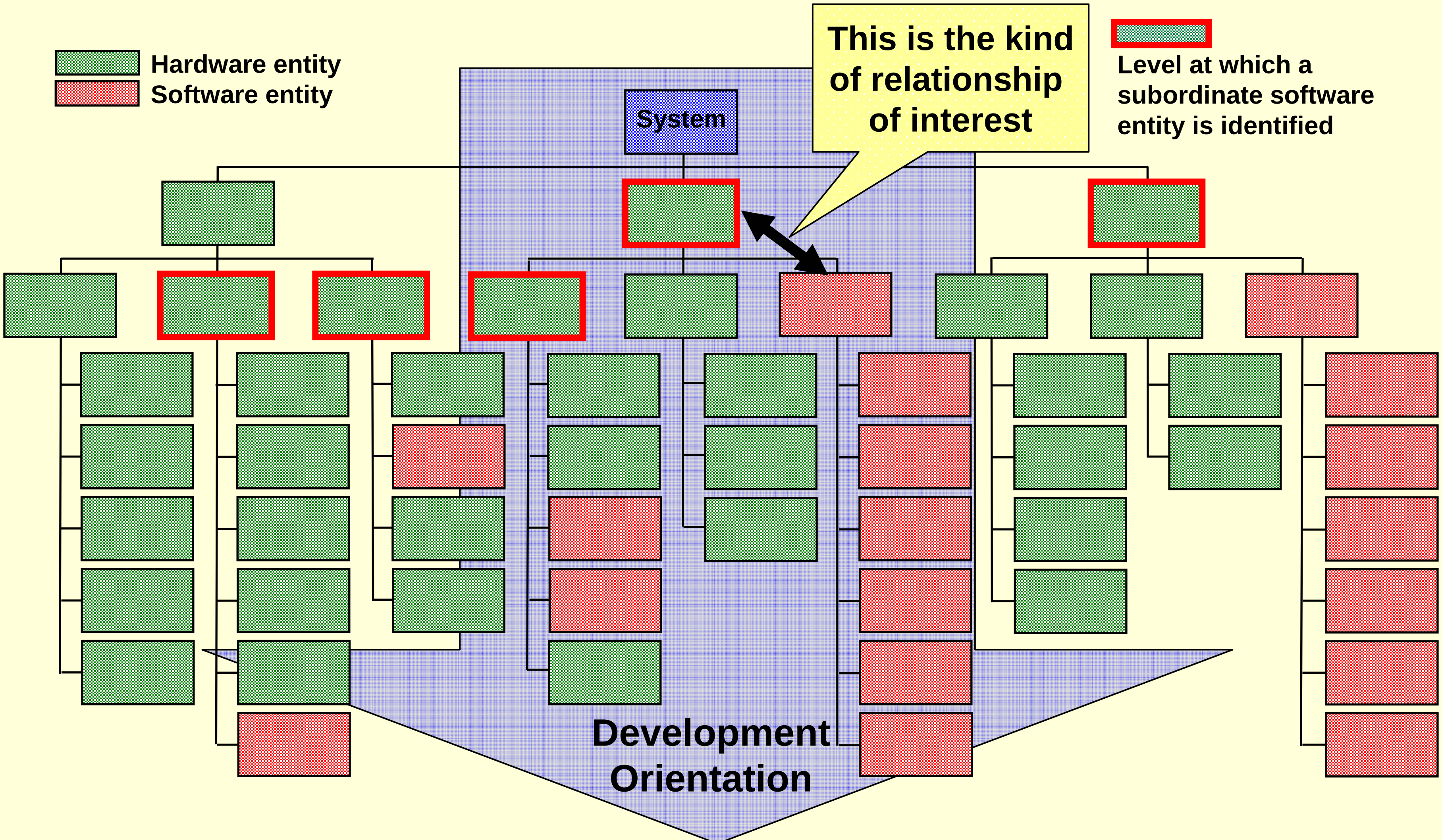
Software Entities

Requirements Traceability Concerns

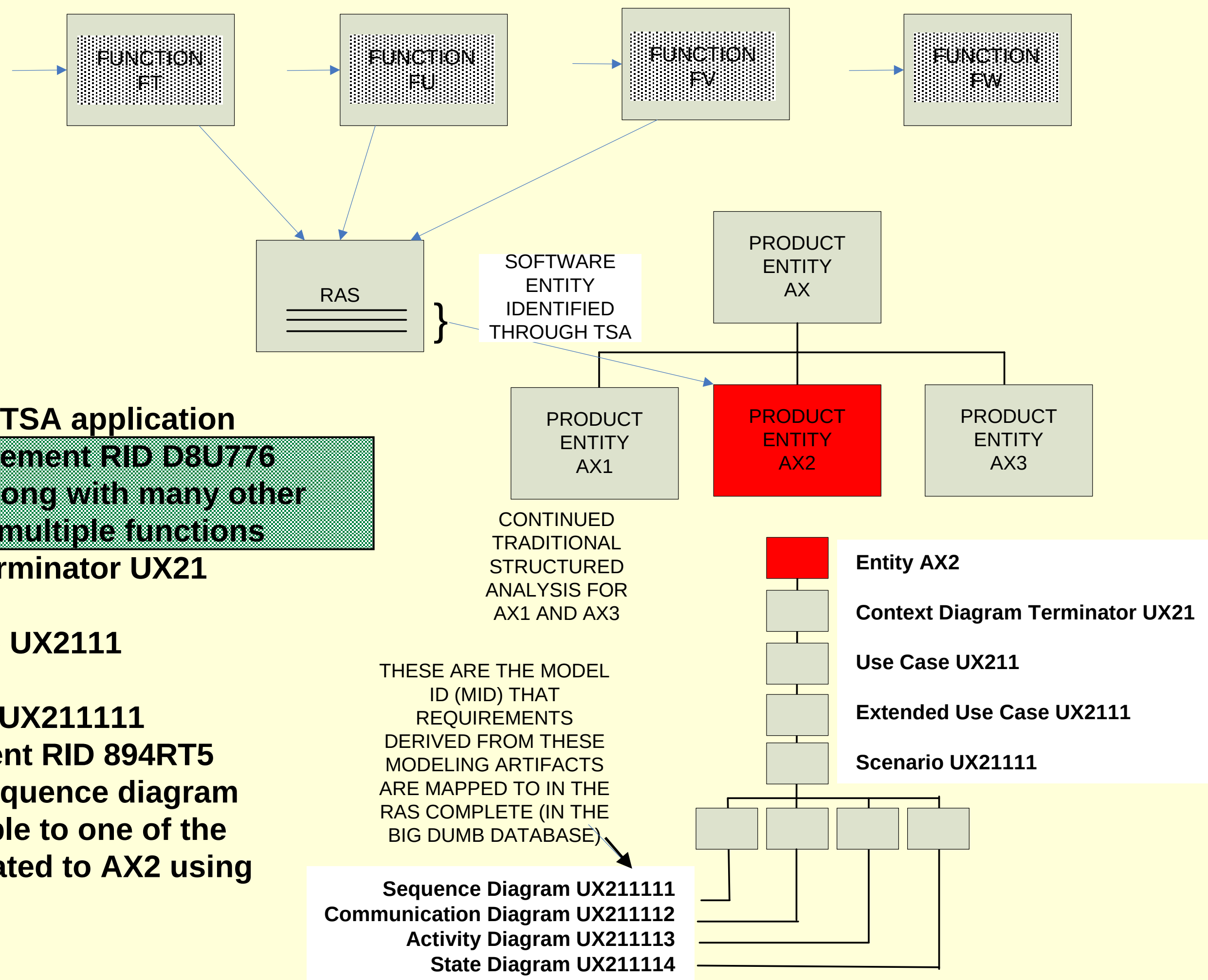
Downward Traceability Situation

		FROM	
		HW	SW
TO	HW	No Problem	Not Going To Happen
	SW	Problem	No Problem

The System Product Entity Structure



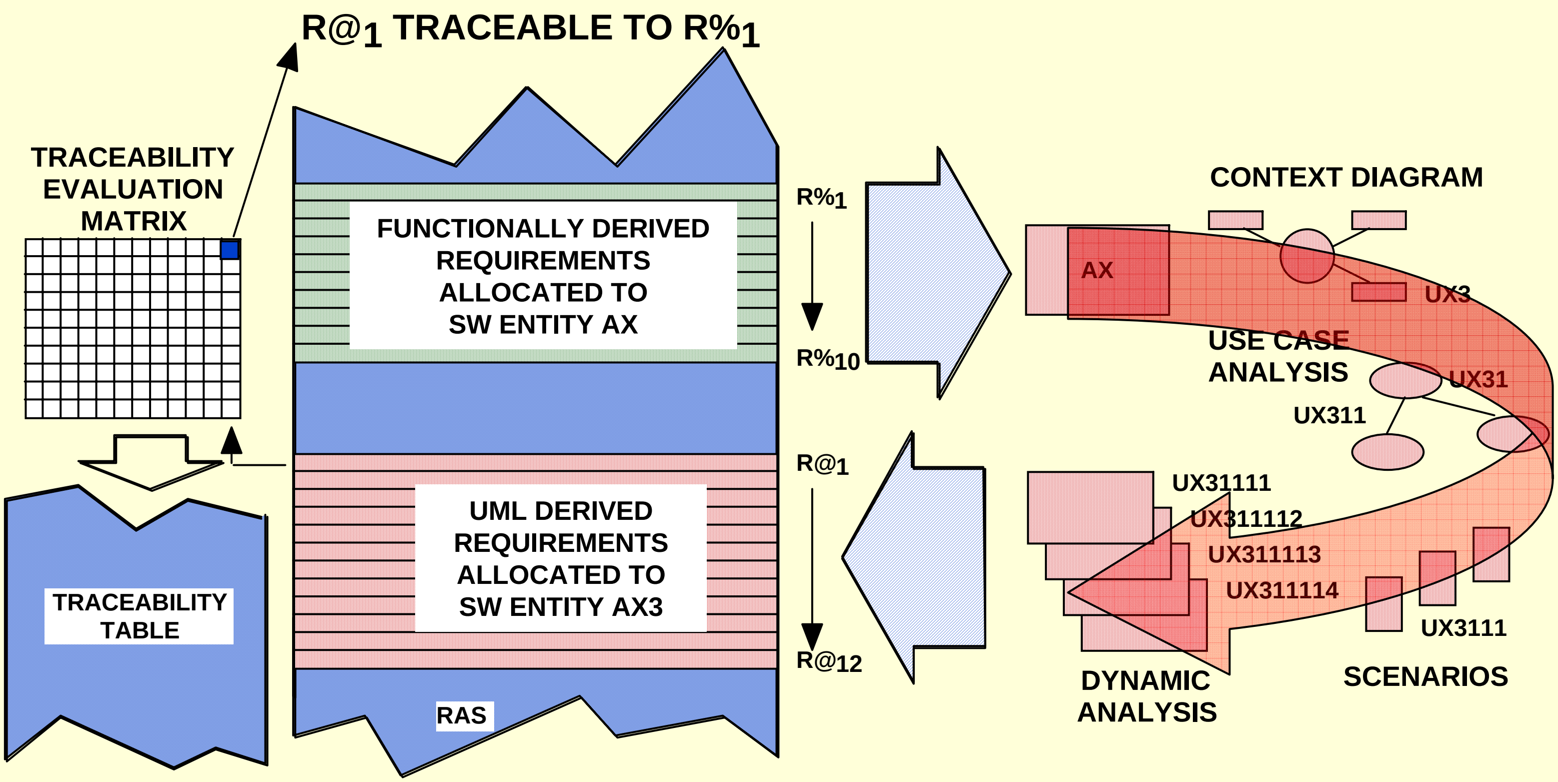
Traceability Across the Gap



The Gap

- **Function FT within TSA application**
- **Performance requirement RID D8U776 allocation to AX2 along with many other requirements from multiple functions**
- **Context diagram terminator UX21**
- **Use case UX211**
- **Extended Use Case UX2111**
- **Scenario UX21111**
- **Sequence diagram UX211111**
- **Software requirement RID 894RT5 derived from the sequence diagram**
- **RID 894RT5 traceable to one of the requirements allocated to AX2 using TSA.**

Traceability Evaluation Matrix



NDIA NATIONAL DEFENSE INDUSTRIAL ASSOCIATION

Traceability Evaluation Matrix

Requirements Derived From UML Modeling (columns)

Requirements Derived From TSA Modeling (rows)

	R@1	R@2	R@3	R@4	R@5	R@6	R@7	R@8	R@9	R@10	R@11	R@12
R%1	■											
R%2			■			■						
R%3	■			■								
R%4												■
R%5					■							
R%6							■				■	
R%7		■		■								
R%8									■			
R%9								■				
R%10										■		

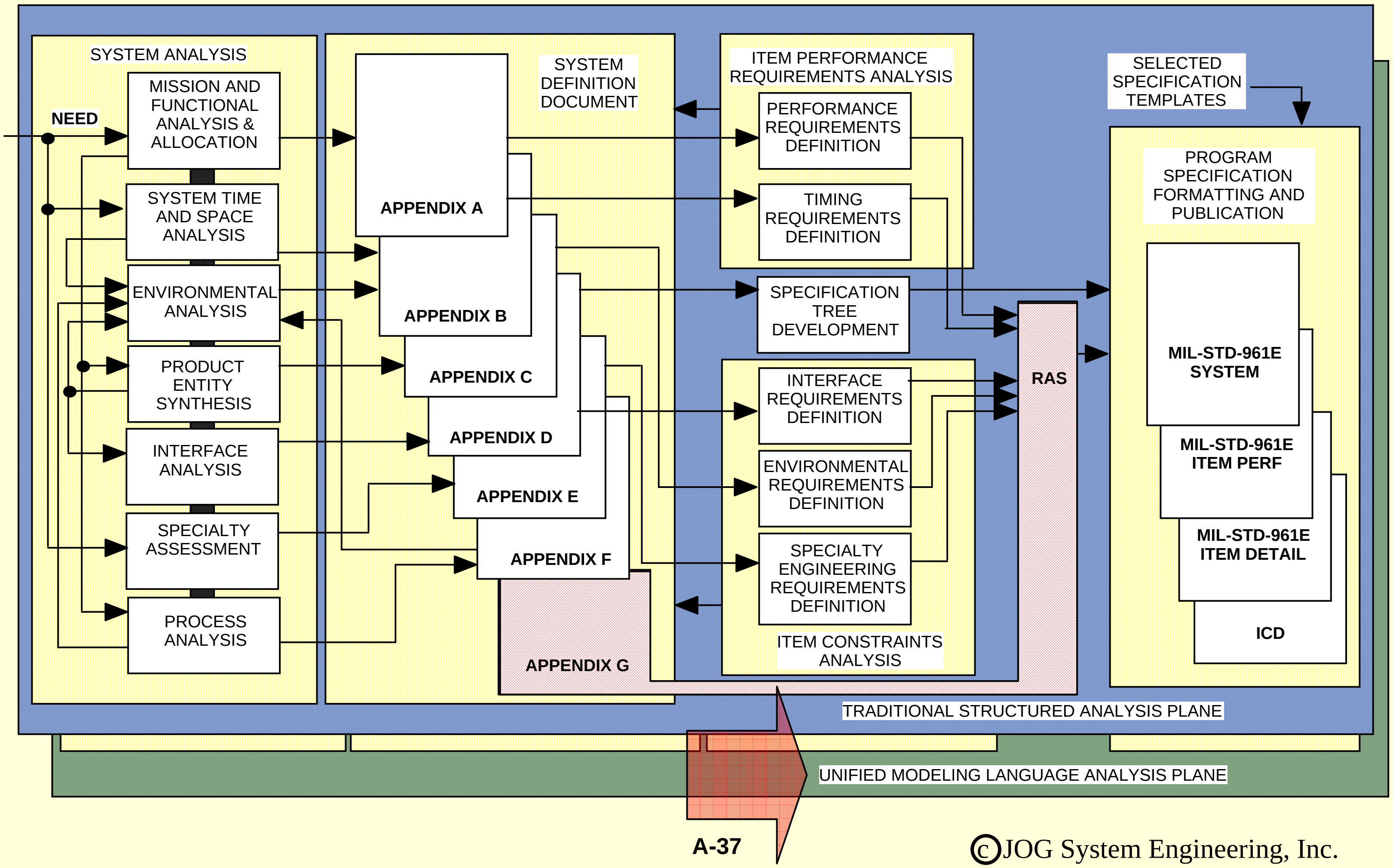
POSSIBLE RID EXAMPLE

R%1	RU7Z7H
R%2	R9IER6
R%3	R937YF
R%4	RJ8E6G
R%5	RJYT6T
R%6	RHGT5T
R%7	RID87W
R%8	RBJ8S7
R%9	RL34DF
R%10	R456HD

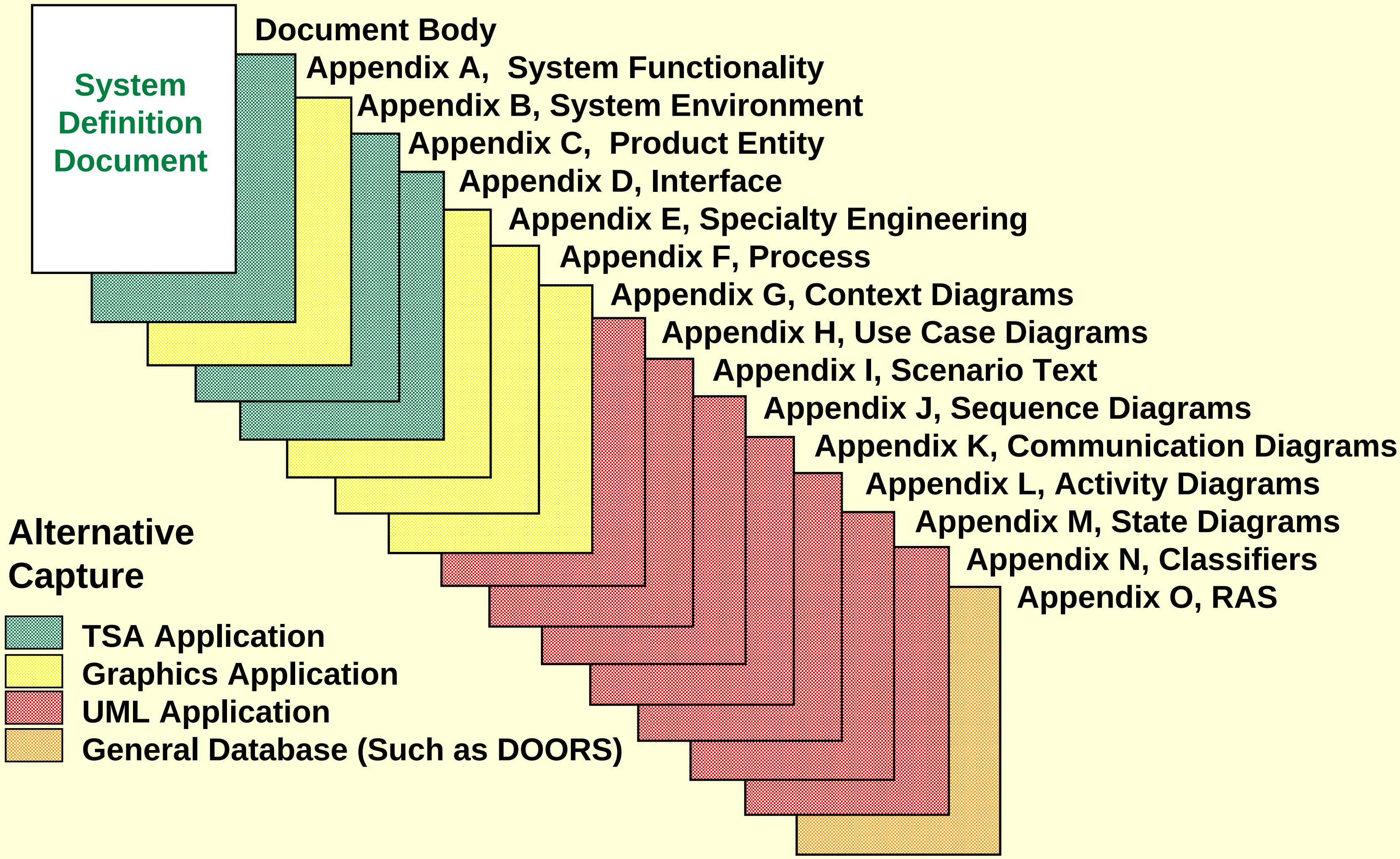
Alternatively, one could rely upon experienced inspection without the organizing influence of the matrix.

NDIA NATIONAL DEFENSE INDUSTRIAL ASSOCIATION

Save the UML Models Too!

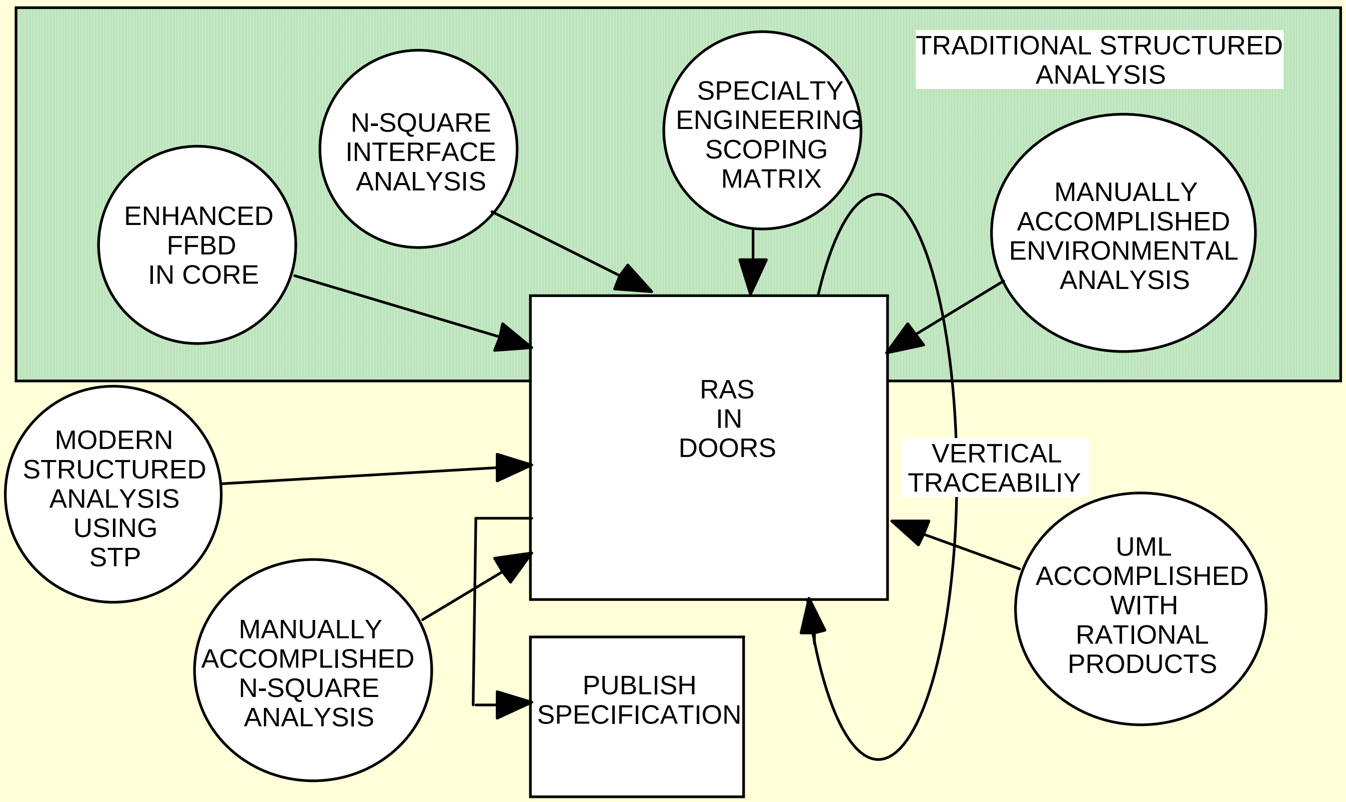


Combined SDD



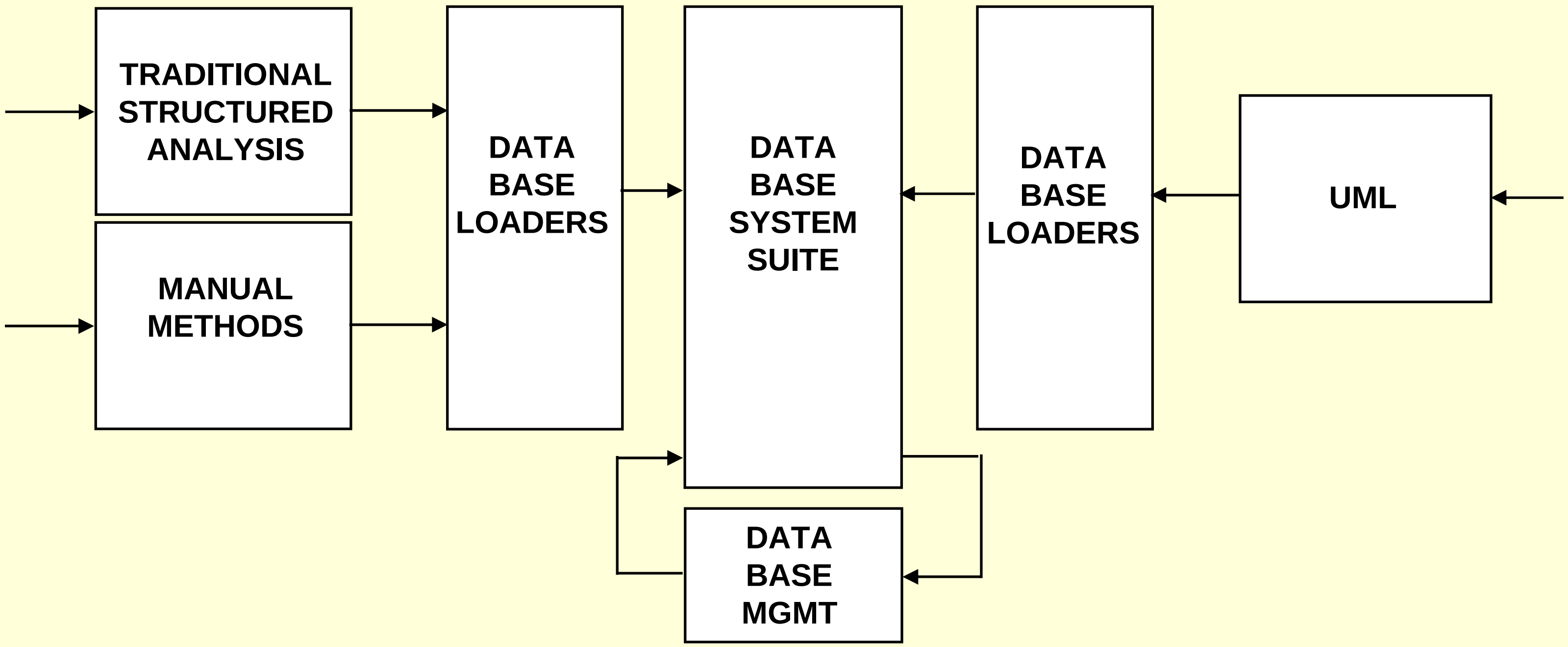
NDIA
NATIONAL DEFENSE INDUSTRIAL ASSOCIATION

A Near Term Tool Set Solution

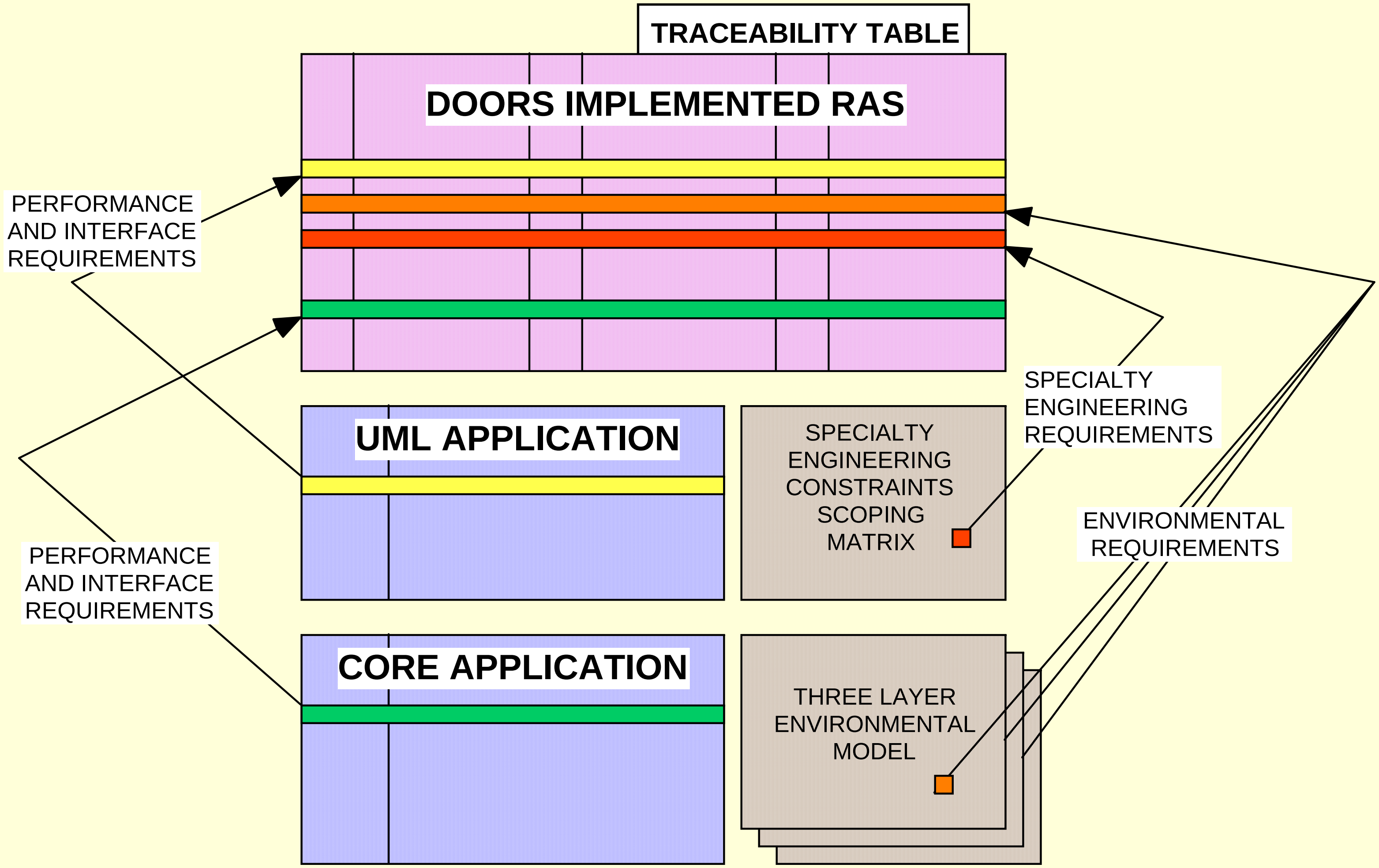


NDIA
NATIONAL DEFENSE INDUSTRIAL ASSOCIATION

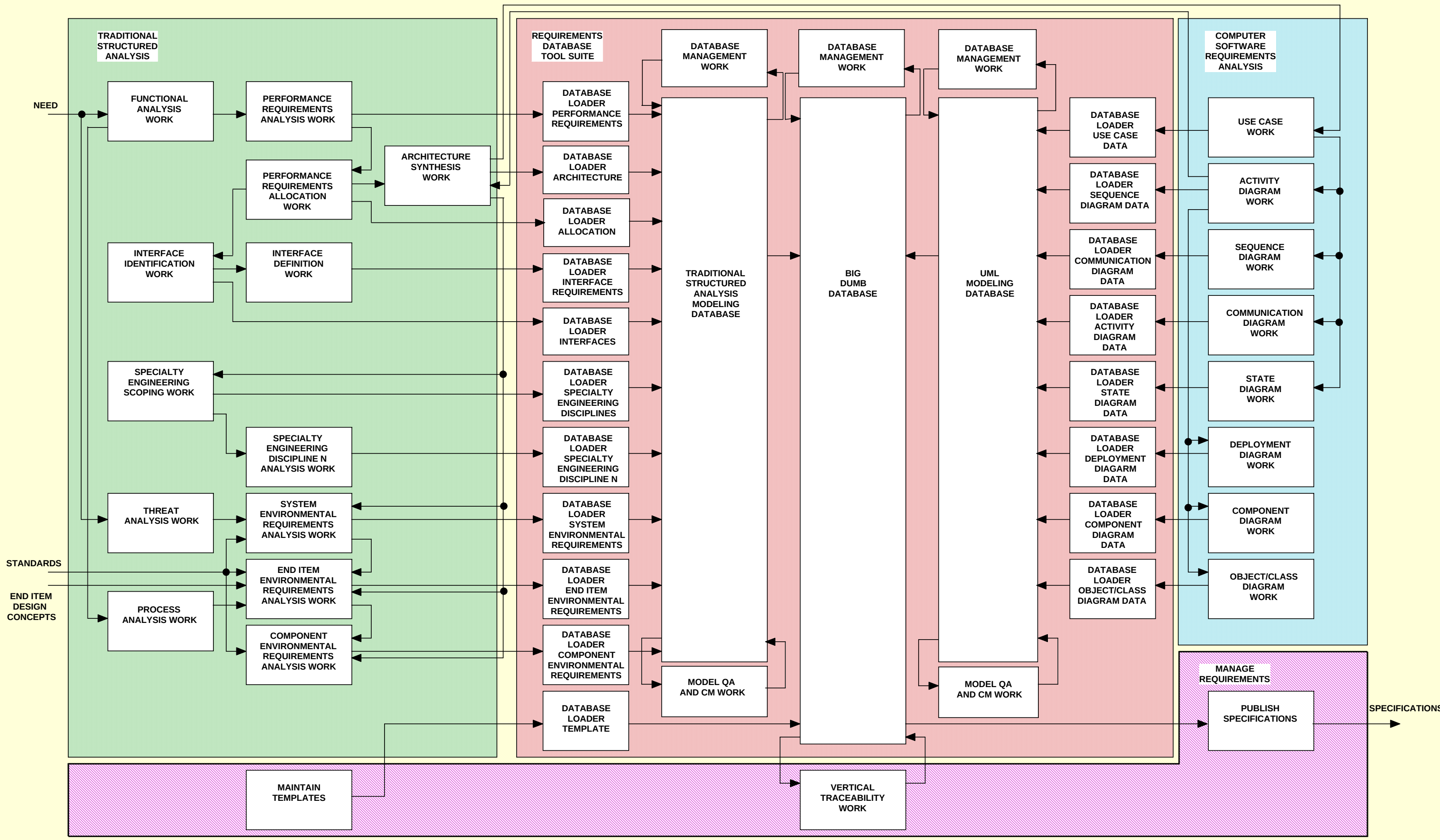
Tools Integration



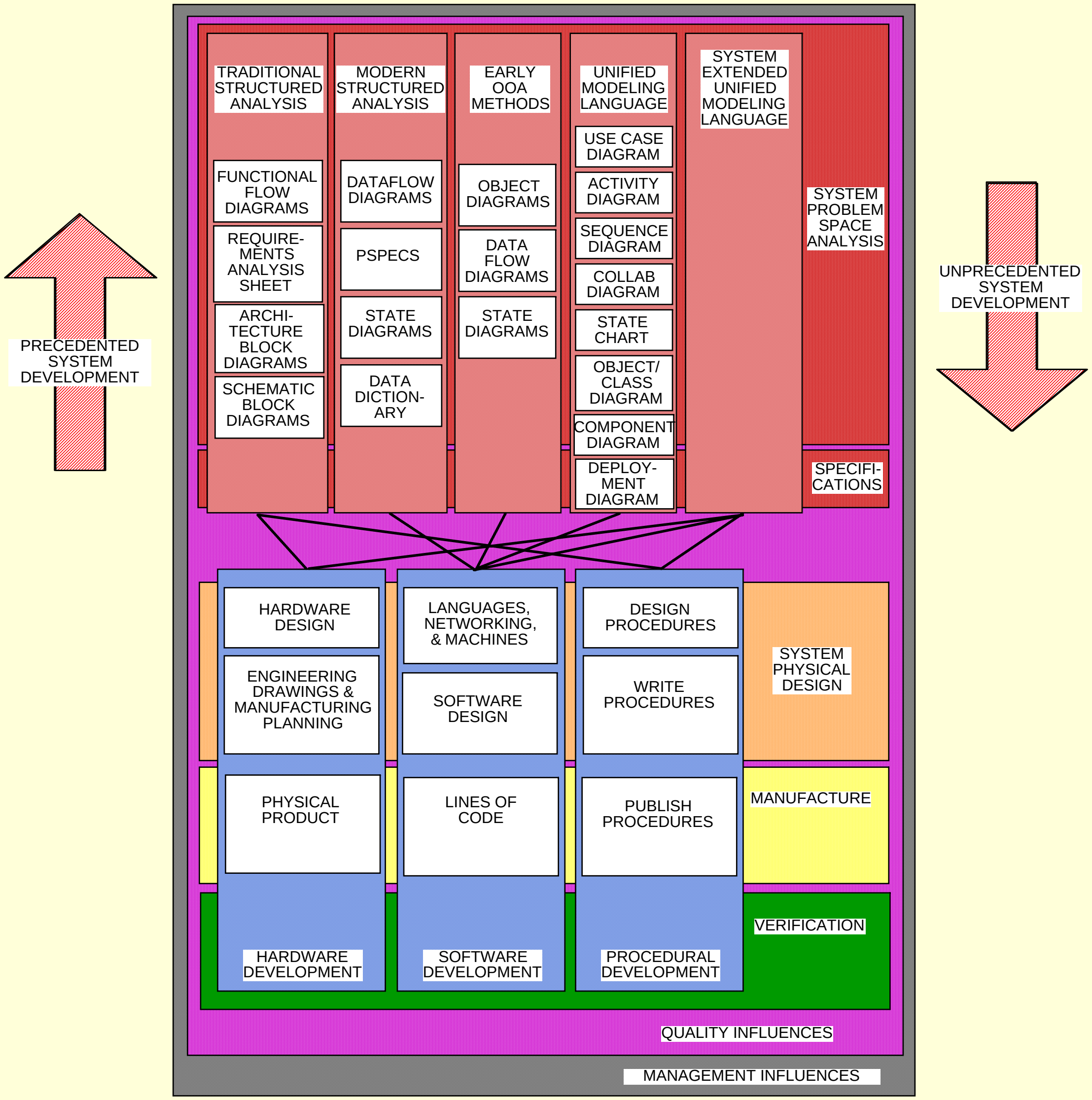
Tools Integration



An Interim Integrated Tool Set



Movement To Universal Method



NDIA NATIONAL DEFENSE INDUSTRIAL ASSOCIATION

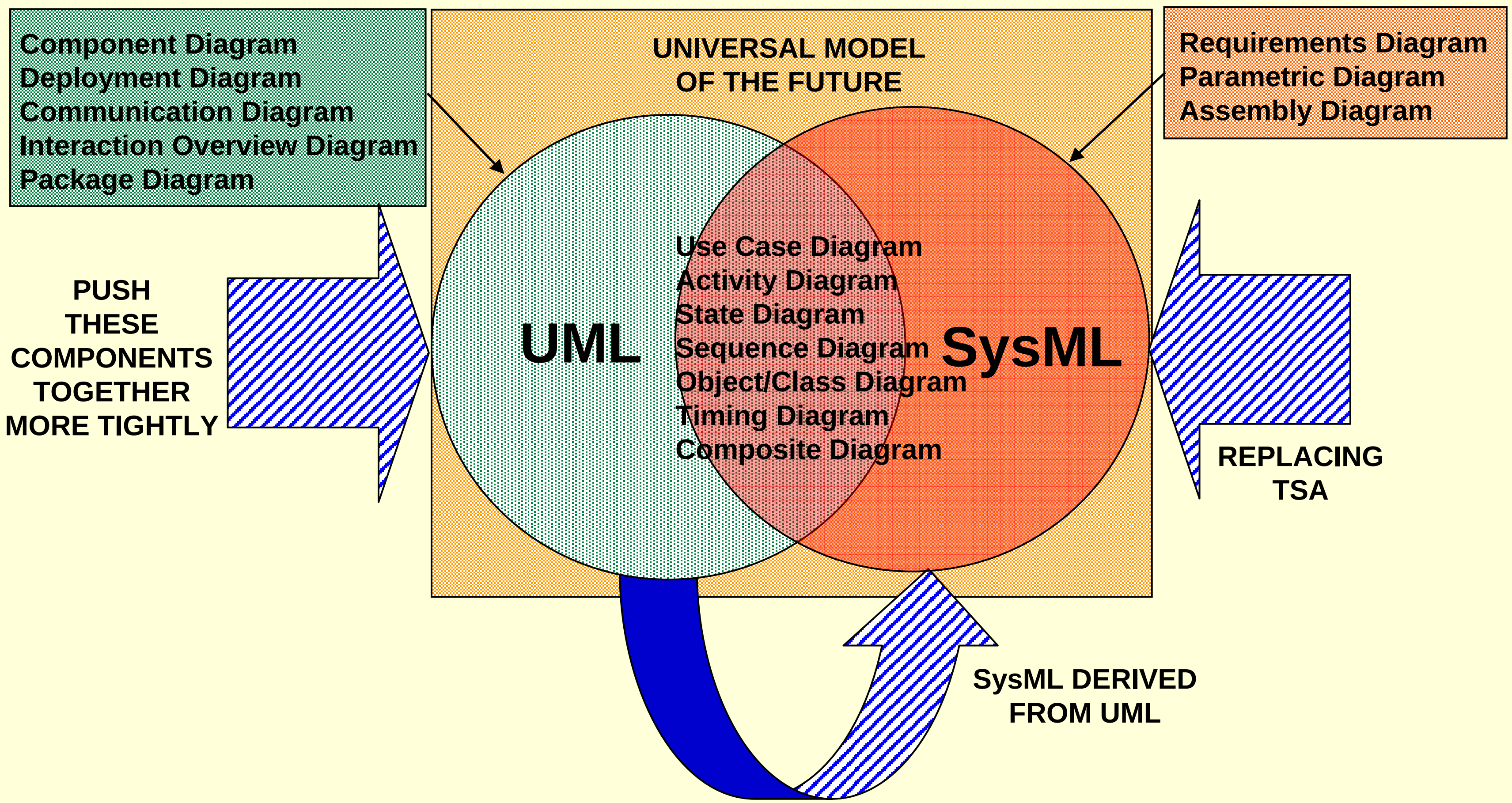
UML and Functional Analysis

Unified Modeling Language (UML)

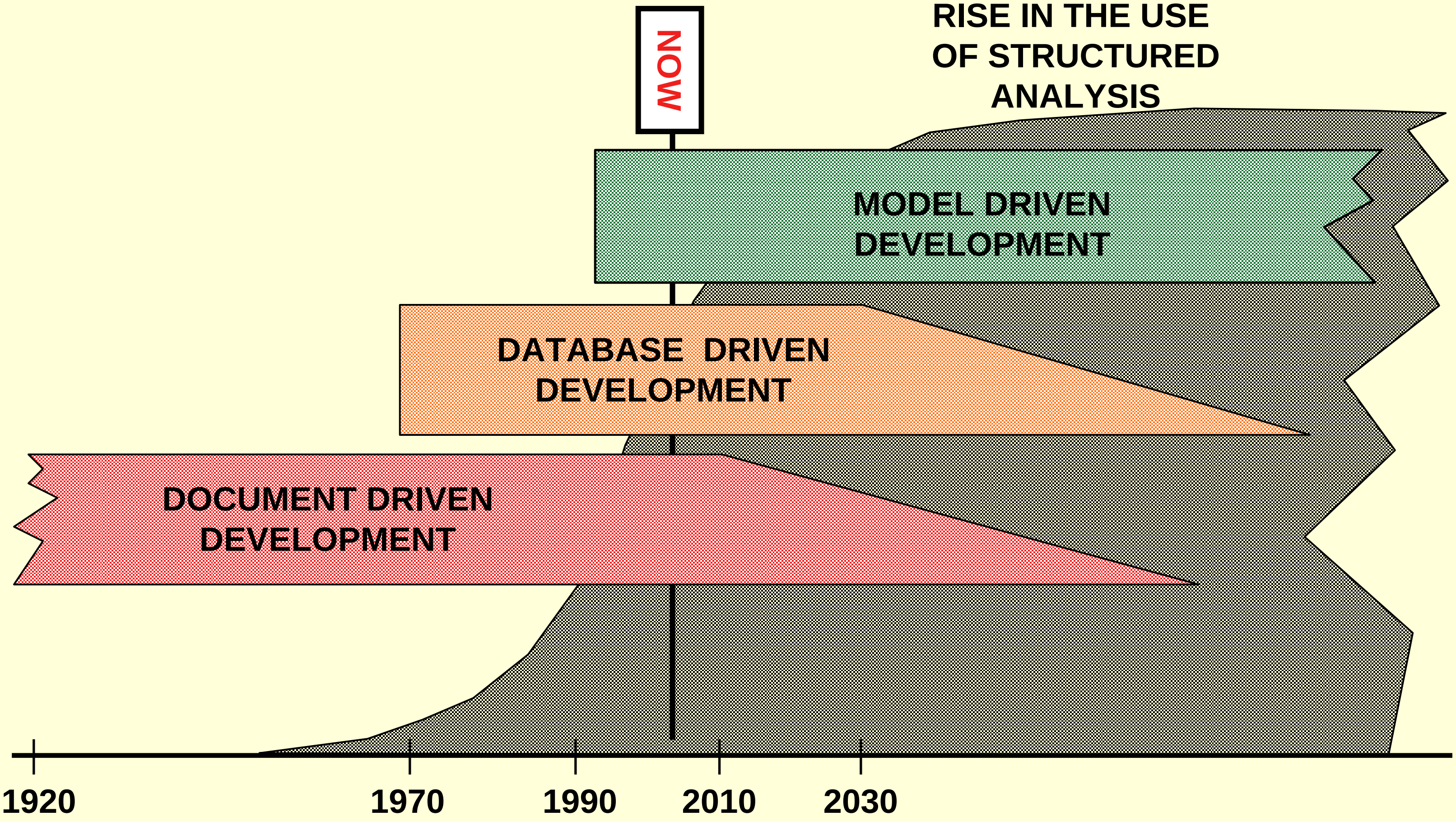
STATIC DIAGRAMS			DYNAMIC DIAGRAMS				
				INTERACTION DIAGRAMS			
DEPLOY-MENT DIAGRAM	COMPONENT DIAGRAM	OBJECT & CLASS DIAGRAMS	STATE CHART	COMMUNI-CATION DIAGRAM	SEQUENCE DIAGRAM	USE CASE DIAGRAM	ACTIVITY DIAGRAM
ARCHITECTURE BLOCK DIAGRAM			STATE DIAGRAM	SCHEMATIC BLOCK DIAGRAM	TIMELINE DIAGRAM	FUNCTIONAL FLOW DIAGRAM	
PHYSICAL FACET			BEHAVIORAL FACET			FUNCTIONAL FACET	

Traditional Structured Analysis
A Subset of UML?

Modeling Changes In the Near Term



System Modeling Evolution Timeline



05-15-2002 DATA UNSUBSTANTIATED
DATES ARE APPROXIMATE

Over the Hill and Through the Woods to Utopia