9th Annual Systems Engineering Conference
“Focusing on Improving Performance of Defense Systems Programs”
San Diego, CA
23-26 October 2006

Table of Contents

Agenda

Monday, 23 October 2006

Track 1:
Practical Six Sigma Tools for Systems Engineering by Dr. Rick Hefner

Track 2
Engineering Workshops by Mr. John Landon
Models for Product Quality by Mr. John Landon

Track 5
Systems Engineering for the Joint Capabilities Integration and Development System (JCIDS) by Mr. Christopher Ryder

Track 6
Grand Systems Development Training Program by Mr. Jeffrey O. Grady
Integrating Systems Engineering with Earned Value Management by Mr. Paul J Solomon

Track 7
Systems Level Configuration Management by Mr. Al Florence
Introduction OMG Systems Modeling Language (OMG SysML™) and OOSEM Tutorial by Dr. Abe Meilich

Track 8
QUality Assessment of System Architectures (QUASAR) by Donald Firesmith
QUality Assessment of System ARchitectures (QUASAR)

Donald Firesmith

Acquisition Support Program (ASP)

Sponsored by the U.S. Department of Defense
© 2006 by Carnegie Mellon University
Topics

What is System Architecture?
Why is System Architecture Critical?
Why Assess the System Architecture?
QUASAR System Architecture Assessment Method:
  • Philosophy
  • Quality Cases
  • QUASAR Process
What is a System Architecture?

Traditional Definition:
the fundamental *structure* of a system in terms of its major components, their relationships to each other and the system’s environment, and the principles governing the creation and evolution of the structure

More General Definition:
the *most important, pervasive, top-level, strategic inventions, decisions, and their associated rationales* about the system including its overall structure (i.e., essential architectural elements, their relationships, and their associated blackbox characteristics and behavior)
Architecture vs. Design

Architecture Decisions:
- Pervasive across System Components
- Strategic Decisions and Inventions
- Higher-Levels of System
- Huge Impact on Quality, Cost, and Schedule
- Drives Design, Highest-Level Integration, and Integration Testing
- Driven by Requirements and Higher-Level Architecture
- Mirrors Top-Level Organization of Development Team

Design Decisions:
- Local within Individual System Components
- Tactical Decisions and Inventions
- Lower-Levels of System
- Smaller Impact on Quality, Cost, and Schedule
- Drives Implementation, Lowest-Level Integration, and Unit Test
- Driven by Requirements, Architecture, and Higher-Level Design
Why is Architecture Critical?

Architecture Defines:
- Key System Components
- How Key Components Interact

Architecture Affects:
- Design Decisions
- Implementation Decisions
- Integration Decisions
- Testing Decisions

Architectural Decisions Drive:
- Ultimate System Quality
- Development Costs
- Development Schedule
- Sustainment Costs
- Maintenance and Upgradeability
Why is Architectures Critical?

The quality of the architecture drives the quality of the system:

- Availability
- Interoperability
- Modifiability
- Performance
- Reliability
- Robustness (Error, Failure, and Fault Tolerance)
- Safety
- Security
- Scalability
- Stability
- Testability
- ...
Why Assess the Architecture?

Determine System Architecture:
  • Quality
  • Maturity and Completeness
  • Integrity and Consistency
  • Usability

Determine Compliance:
  • Contract Compliance
  • Requirements Compliance

Early Identification of System Architecture Defects:
  • Fix Defects Early
  • Decrease Costs
  • Decrease Schedule
Why Assess the Architecture?

Manage Risks:
- System Architecture Risks
- System Risks

Provide Acquirer Oversight into System Architecture

Develop Consensus:
- Among Developers
- Between Acquirer and Developer Organization

Ensure Specification of Quality Requirements

Help Architects Succeed

Help Program Succeed
How to Assess the Architecture?

Assessment Philosophy
Quality Cases as Foundation
QUASAR Process:
  • Phases
    - System Architecture Assessment Initiation
    - Subsystem Requirements Review
    - Subsystem Architecture Assessment
    - System Architecture Assessment Summary
Assessment Philosophy

Quality Requirements Drive the System Architecture.
Architects should Make Case to Assessors:
• Architects Know Quality Requirement Drivers
• Architects Know What they Did and Why
• Architects Know Where Documented

Safety Cases can Generalize into Quality Cases
(a.k.a., assurance cases) consisting of:
• Claims: Architecture Supports Quality Requirements
• Arguments: Architects’ Architectural Decisions and Rationales
• Evidence: Architects’ Documentation and Witnessed Demonstrations
Assessment Philosophy

Arguments must be Clear and Compelling.
Evidence must Be Credible.

Architects’ Responsibilities:
- Prepare Quality Cases
- Provide Early Presentation Materials to Assessors
- Present Quality Cases (Make Case to Assessors)
- Answer Assessors’ Questions

Assessor Responsibilities:
- Prepare for Assessments
- Probe Quality Cases
- Determine and Report Assessment Results
Quality Cases – Quality Model

Quality of a system (and system architecture) is defined in terms of a quality model:

- Quality Model defines the meaning of quality for the system.
- Quality Factor defines a type of the quality of the system.
- Quality Subfactor defines a part of a type of the quality of the system.
- Quality Measure (Measurement Scale) is measured using a scale.

System

Quality Model —— 1..* —— Quality Factor —— 1..* —— Quality Subfactor —— 0..* —— Quality Measure (Measurement Scale)
Quality Cases – Quality Factors

- Quality Factor
  - Development-Oriented Quality Factor
  - Usage-Oriented Quality Factor

- Quality Subfactor
  - Soundness
  - Correctness
  - Predictability
  - Operational Availability
  - Reliability
  - Stability
  - Dependability
    - Efficiency
    - Interoperability
    - Performance
    - Utility
  - Security
    - Robustness
  - Safety
    - Soundness
  - Capacity
  - Configurability

Quality Model

is measured using a

Quality Measure (Measurement Scale)
Quality Cases – Quality Subfactors

- Harm
- Accident & Safety Incident
- Nonmalicious Agent
- Internal Vulnerability
- Hazard
- Safety Risk

Safety Problem Type

Safety Solution Type

Prevention
Detection
Reaction
Adaptation

Safety
Safety Subfactor
Quality Cases - Components

1. Claims
   Their architecture adequately supports its derived and allocated quality goals and requirements

2. Clear and compelling Arguments
   • Architecture decisions
   • Associated rationales

3. Supporting Evidence
   • Official program documentation
   • Witnessed demonstrations

Simplified version of safety case from safety community
Quality Cases - Relationships

Quality Subfactor

Quality Factor

defines a part of a type of quality of a

is specific to a

Quality Case

makes the case for the quality of a

System

Subsystem

defines a type of quality of a

Claim

justifies belief in

Argument

supports

Evidence

System

Subsystem

Version 0.2

© 2006 by Carnegie Mellon University
Architecture Quality Cases

Architecture Quality Case

Quality Case

Claim

Argument

Evidence

Architecture

System

Subsystem

Architectural Claim

Architectural Argument

Architectural Evidence

justifies belief in

supports

justifies belief in

supports

makes the case for the quality of an

has an

Architectural Claim

Architectural Argument

Architectural Evidence

 Claim

Argument

Evidence

Architectural Claim

Architectural Argument

Architectural Evidence

justifies belief in

supports

Architectural Claim

Architectural Argument

Architectural Evidence

justifies belief in

supports

makes the case for the quality of an

has an

Architecture

System

Subsystem
Architecture Quality Case

Architecture Quality Case makes architects’ case for the quality of an architecture.

Architecture Claim justifies belief in the quality of an architecture.

Architecture Argument supports the architecture claim.

Architecture Evidence concerns the architecture claim.

Quality Goal Claim claims the architecture adequately helps the system achieve its goals.

Quality Requirement Claim claims the architecture adequately helps the system meet its requirements.

Quality Goal ensures achieving the quality of the system.

Quality-Related Requirement supports the quality goal.

Architectural Decision justifies the quality requirement.

Rationale supports the architectural decision.

Quality Factor Requirement ensures meeting the quality factor.

Quality Subfactor Requirement ensures meeting the quality subfactor.

Quality Constraint ensures meeting the quality constraint.

Official Documentation (e.g., Diagrams, Models, and Documents)

Witnessed Demonstrations (e.g., Scenarios, Tests, and Simulations)
Interoperability (Quality) Case
Quality Case Diagram

Quality Cases contain a large amount of Information. Claims, Arguments, and a large amount of Evidence are typically text. It is easy to get lost in a large, complex, textual quality case. A quality Case Diagram is a layered UML class diagram that labels and summarizes the parts of a single quality case:

- **Claims:**
  - Quality Goals
  - Quality Requirements
- **Arguments:**
  - Architectural Decisions
  - Rationale
- **Evidence:**
  - Documentation
  - Witnessed Demonstrations
Interoperability Quality Case Diagram

**Goal:**
Architecture Supports Interoperability
<<claim>>

- **Goal:**
  Architecture Supports Physical Interoperability
  <<claim>>

**Requirements:**
Architecture Supports Physical Interoperability
<<claim>>

**Architecture Decision:**
One-Way Connections
<<argument>>

**Architecture Decision:**
Layered Architecture
<<argument>>

**Architecture Decision:**
Open Interface Standards
<<argument>>

**Architecture Decision:**
Service Oriented Architecture
<<argument>>

**Architecture Decision:**
Fly-By-Wire
<<argument>>

**Architecture Decision:**
Modular Architecture
<<argument>>

**Architecture Decision:**
Proxies and Wrappers
<<argument>>

**Wiring Diagram**
<<evidence>>

**Context Diagram**
<<evidence>>

**Allocation Diagram**
<<evidence>>

**Layer Diagram**
<<evidence>>

**Interoperability Whitepaper**
<<evidence>>

**Vendor-Supplied Technical Documentation**
<<evidence>>

**Hardware Schematics**
<<evidence>>

**Configuration Diagram**
<<evidence>>

**Network Diagrams**
<<evidence>>

**Activity or Collaboration Diagrams**
<<evidence>>

© 2006 by Carnegie Mellon University
Version 0.2
Partial Performance Quality Case
Diagram

Goal: Architecture Supports Performance <<claim>>

Goal: Architecture Limits Jitter <<claim>>

Goal: Architecture Limits Latency <<claim>>

Goal: Architecture Limits Response Time <<claim>>

Goal: Architecture Limits Latency <<claim>>

Goal: Architecture Supports Throughput <<claim>>

Requirements: Architecture Limits Jitter <<claim>>

Requirements: Architecture Supports Schedulability <<claim>>

Requirements: Architecture Limits Response Time <<claim>>

Requirements: Architecture Limits Latency <<claim>>

Requirements: Architecture Supports Throughput <<claim>>

justifies belief in

Architecture Decision: Real-Time Operating System <<argument>>

Architecture Decision: Deterministic Scheduling <<argument>>

Architecture Decision: Layered Architecture <<argument>>

Architecture Decision: Redundant Hardware <<argument>>

Architecture Decision: Load Balancing <<argument>>
QUASAR Assessment Process

Four Phases:
1. System Architecture Assessment Initiation (SAAI)
   For each Subsystem to be assessed:
2. Subsystem Requirements Review (SRR)
3. Subsystem Architecture Assessment (SAA)
4. System Architecture Assessment Summary (SAAS)

Each Phase consists of 3 Tasks:
1. Preparation
2. Meeting
3. Follow-Through
QUASAR Phases

System Architecture Assessment Initiation

Subsystem Requirements Review

Subsystem Architecture Assessment

System Architecture Assessment Summary

repeat for each subsystem being assessed

no

done

yes
QUASAR Phases and Tasks

System Architecture Assessment Initiation Phase

<table>
<thead>
<tr>
<th>Prep.</th>
<th>Initial Meeting</th>
<th>Follow Through</th>
</tr>
</thead>
</table>

Subsystem 1 Architecture Assessment

Subsystem Requirements Review Phase

Prep. | Rqmts. Meeting | Follow Through |

Subsystem Architecture Assessment Phase

Prep. | Arch. Meeting | Follow Through |

Subsystem 2 Architecture Assessment

Subsystem Requirements Review Phase

Prep. | Rqmts. Meeting | Follow Through |

Subsystem Architecture Assessment Phase

Prep. | Arch. Meeting | Follow Through |

Subsystem N Architecture Assessment

Subsystem Requirements Review Phase

Prep. | Rqmts. Meeting | Follow Through |

Subsystem Architecture Assessment Phase

Prep. | Arch. Meeting | Follow Through |

System Architecture Assessment Summary Phase

Prep. | Final Meeting | Follow Through |

Time (not to scale) →
Quasar Teams

- **Requirements Team(s)**
- **Architecturally Significant (Quality) Requirements**
- **Architecture**
- **Subsystem Architectures**
- **System Architecture**
- **Top-Level Architecture Team**
- **Subsystem Architecture Teams**
- **Quality Cases**
- **Assessment Team(s)**

- Produce the
- Drive the
- Assess the quality of the
- Produce the
- Produces the
- Leads the
- Make their
- Make their
- Evaluate the architects'
System Architecture Assessment Initiation (SAAI) Phase

- System Architecture Assessment Initiation
  - Subsystem Requirements Review
  - Subsystem Architecture Assessment
    - repeat for each subsystem being assessed
    - no
      - done
    - yes
      - System Architecture Assessment Summary
SAAI Topics

SAAI Phase Objectives
SAAI Phase Principles
SAAI Phase Context
SAAI Phase Overview
  • SAAI Preparation Task
  • SAAI Meeting Task
  • SAAI Follow-Through Task
SAAI Roles and Responsibilities
Discussion
SAAI Phase Objectives

Prepare the teams

Develop Consensus:
  • Scope the Assessments
  • Schedule the Assessments
  • Tailor the Assessment Process and Training Materials
  • Capture Lessons Learned

Produce and Publish Meeting Outbrief and Minutes

Manage Action Items
SAAI Phase Principles

Need to Develop Consensus between Assessors and Assesses
Need to Tailor Process to meet specific Needs of the Overall Assessment
Scope of Assessment should match Project Needs and Resources
Subsystem Assessments must be scheduled to ensure required Resources
SAAI Phase Context

System Architecture Assessment Initiation Phase
Prep. Initial Meeting Follow Through

Subsystem 1 Architecture Assessment
Subsystem Requirements Review Phase
Prep. Rqmts. Meeting Follow Through

Subsystem Architecture Assessment Phase
Prep. Arch. Meeting Follow Through

Subsystem 2 Architecture Assessment
Subsystem Requirements Review Phase
Prep. Rqmts. Meeting Follow Through

Subsystem Architecture Assessment Phase
Prep. Arch. Meeting Follow Through

Subsystem N Architecture Assessment
Subsystem Requirements Review Phase
Prep. Rqmts. Meeting Follow Through

Subsystem Architecture Assessment Phase
Prep. Arch. Meeting Follow Through

System Architecture Assessment Summary Phase
Prep. Final Meeting Follow Through

Time (not to scale) →
SAAI Phase Overview

System Architecture Assessment Initiation Phase

Preparation Task
- Architecture Assessment Training Materials
- Architecture Assessment Procedure

Meeting Task
- Initial Kick-off Meeting Agenda
- Initial Kick-off Meeting Notes
- Architecture Assessment Schedule
- Architecture Assessment Action Item List
- Initial Kick-off Meeting Minutes

Follow-Through Task

Assessment Team
- discussions

Top-Level Development Team
- discussions

Subsystem Requirements Teams
- discussions

Top-Level Requirements Team
- discussions

Top-Level Architecture Team
- discussions

Subsystem Architecture Teams
- discussions

© 2006 by Carnegie Mellon University
Version 0.2
QUASAR Method. - page 32
SAAI Preparation Task

Steps:

1. Staff Assessment Team
2. Train Assessment Team
3. Assessment Team Identifies the Top-Level Development Team (Top-Level Requirements Engineers & Architects)
4. Assessment Team Trains Top-Level Development Team
5. Teams Collaborate to Organize Meeting (attendees, time, location, agenda)
SAAI Meeting Task

Steps:

1. Teams Collaborate to Determine Assessment Scope:
   - Architecturally Significant Requirements
   - Subsystems
   - Assessment Resources

2. Teams Collaborate to Develop Initial Assessment Schedule

3. Teams Collaborate to Tailor Assessment Process

4. Assessment Team Manages Action Items
SAAI Follow-Through Task

Steps:
1. Assessment Team develops and presents Meeting Outbrief
2. Assessment Team develops, reviews, and distributes Meeting Minutes
3. Assessment Team tailors and distributes:
   - Assessment Procedure
   - Assessment Training Material
4. Assessment Team distributes Assessment Schedule
5. Teams obtain Needed Resources
6. Assessment Team captures Lessons Learned
7. Assessment Team Manages Action Items
SAAI Roles and Responsibilities

The system architecture assessment initiation phase is performed by the following teams:

- Assessment Team
- Top-Level Development Team:
  - Top-Level Requirements Team with input from Subsystem Requirements Teams
  - Top-Level Architecture Team with input from Subsystem Requirements Teams
Assessment Team Membership

Assessment Team Leader
Assessors
Meeting Facilitator
Subsystem Liaison
Subject Matter Experts
Scribe
Assessment Team Responsibilities

Provide Architecture Assessment Training Materials
Provide Architecture Assessment Procedure
Collaborate to Tailor Architecture Assessment Procedure
Collaborate to provide Initial Kick-Off Meeting Agenda
Take Initial Kick-Off Meeting Notes
Collaborate to Develop Assessment Schedule
Produce Architecture Assessment Action Item List
Produce Outbrief and Meeting Minutes
Development Team Membership

Top-Level Requirements Team:
• Lead Requirements Engineer
• System Requirements Engineers
• Subsystem Requirements Engineers

Top-Level Architecture Team:
• Lead System Architect
• System Architects
• Subsystem Architects
Development Team Responsibilities

Read Architecture Assessment Training Materials
Read Architecture Assessment Procedure
Collaborate to Tailor Architecture Assessment Procedure
Collaborate to provide Initial Kick-Off Meeting Agenda
Take Initial Kick-Off Meeting Notes
Collaborate to Develop Assessment Schedule
SAAI Discussion

What is the main objectives of the system architecture assessment initiation phase?

What are the three tasks comprising the SAAI phase?

What teams are involved?

What are the memberships and responsibilities of these teams?
Subsystem Requirements Review (SRR) Phase

1. System Architecture Assessment Initiation
2. Subsystem Requirements Review
   - repeat for each subsystem being assessed
3. Subsystem Architecture Assessment
   - yes
   - System Architecture Assessment Summary
4. no
   - done
SRR Topics

SRR Questions for the Attendees
SRR Phase Objectives
SRR Phase Principles
SRR Phase Challenges
SRR Phase Context
SRR Phase Overview
  •  SRR Preparation Task
  •  SRR Meeting Task
  •  SRR Follow-Through Task
SRR Roles and Responsibilities
Discussion
SRR Questions for the Attendees

As a requirements engineer, what are your biggest problems with respect to engineering (e.g., identifying, deriving, analyzing, specifying, and managing) system and subsystem requirements that significantly impact the architecture?

As a system architect, what are your biggest problems with respect to the system requirements that significantly impact the architecture?

As a subsystem architect, what are your biggest problems with respect to the derived architecturally significant requirements that have been allocated to your subsystem?
Key Questions for the Attendees

How can you know the architecture is ‘good enough’ if the requirements do not specify exactly how good it has to be?

- How else can the *architects*:  
  - Make engineering trade-offs among the different quality factors?  
  - Know when the architecture is done?
- How can the architecture *assessors* assess the quality of the architecture without having requirements against which to make the assessment?
- How can *testers* determine success or failure without measurable test completion criteria?
- How can *managers* know the quality and status of the architecture without measurable indicators?
SRR Phase Objectives

Ensure that the:

1. Architecturally significant requirements are properly engineered in time to support the engineering of the subsystem architecture.

2. Subsystem architects know how to prepare for and support the coming subsystem architecture quality assessment.
Architecturally Significant Requirements

Architecturally Significant Requirement
Any requirement that has a significant impact on the system architecture

Quality requirements are typically the most important architecturally significant requirements.

Definition
Any requirement that specifies a minimum level of quality
Quality Requirements

Format
The system shall do X with threshold Y under condition(s) Z.

Bad Example(s)
The system shall be highly reliable, robust, safe, secure, stable, etc.

Good Example (Stability)
The system shall ensure that the mean time between the failure of non-critical functionality* causing the failure of critical functionality* is least 5,000 hours of continuous operation under normal operating conditions*.

* Must be properly defined in the project glossary.
SRR Phase Principles

Not all requirements are architecturally significant. Quality requirements should be major drivers of the system architecture. Quality requirements should specify a minimum required amount of some type of quality.

Quality requirements should be:
- Unambiguous
- Feasible
- Complete
- Consistent
- Mandatory
- Verifiable
- Validatable
SRR Phase Principles

Quality requirements should be organized according to a quality model that defines quality factors (a.k.a., attributes, “ilities”) and their quality subfactors:

- Availability
- Interoperability
- Performance
  - Jitter, Response Time, Schedulability, and Throughput
- Portability
- Reliability
- Safety
- Security
- Usability
SRR Phase Principles

Different quality factors are important for different subsystems.
  • Performance is paramount for some subsystems.
  • Security is more important for other subsystems.

Engineering architecturally significant requirements is the responsibility of the requirements team, not the architecture team and not the assessment team.
  • Architects and assessors are not qualified to engineer quality requirements.
  • Many stakeholders need quality requirements.
  • Architecture assessment time is too late to engineer quality requirements.
SRR Phase Challenges

Architects are rarely given/allocated a *complete* set of architecturally significant requirements. These architecturally significant requirements rarely include quality requirements for *all* of the relevant quality factors and subfactors.
SRR Phase Challenges

The quality of the derived and allocated architecturally significant requirements are typically poor:

- Requirements are often *ambiguous*.
  - “The system shall be safe and secure.”
- Requirements rarely specify *thresholds* on relevant quality measurement scales.
  - “The system shall have adequate availability.”
- Requirements are often mutually *inconsistent*.
  - Security vs. usability, performance vs. reliability.
- Many requirements are *infeasible* (or at least impractical) if taken literally.
  - “The system shall have 99.999999 reliability.”
SRR Phase Challenges

Requirements are often unstable.
Specialty engineering requirements (e.g., reliability, safety, security) are often documented separately from the functional requirements.
The architecturally significant requirements are often *improperly* prioritized for implementation.
The subsystem architects often do not understand how to prepare for an architecture assessment.
  - Too busy
  - Not trained
  - No standards exist
  - Bias against assessments/audits
SRR Phase Challenges

The subsystem architects do not understand how to give the assessment team the information they need to assess the architecture:

- How good must the architecture be to sufficiently supports its derived and allocated quality requirements (i.e., to ‘pass’ the assessment)?
- What architectural decisions did the architects make to support the quality goals and requirements?
- What were the rationales for these decisions?
- What is the official documentation of actual architectural decisions?
  - Not plans and procedures
  - Official program documentation
  - Not hastily produced PowerPoint slides
SRR Phase Context

System Architecture Assessment Initiation Phase

<table>
<thead>
<tr>
<th>Prep.</th>
<th>Initial Meeting</th>
<th>Follow Through</th>
</tr>
</thead>
</table>

Subsystem Architecture Assessment Phase

<table>
<thead>
<tr>
<th>Subsystem 1</th>
<th>Subsystem 2</th>
<th>Subsystem N</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prep.</td>
<td>Rqmts. Meeting</td>
<td>Follow Through</td>
</tr>
<tr>
<td>Prep.</td>
<td>Rqmts. Meeting</td>
<td>Follow Through</td>
</tr>
<tr>
<td>Prep.</td>
<td>Rqmts. Meeting</td>
<td>Follow Through</td>
</tr>
</tbody>
</table>

Subsystem Requirements Review Phase

<table>
<thead>
<tr>
<th>Subsystem 1</th>
<th>Subsystem 2</th>
<th>Subsystem N</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prep.</td>
<td>Rqmts. Meeting</td>
<td>Follow Through</td>
</tr>
<tr>
<td>Prep.</td>
<td>Rqmts. Meeting</td>
<td>Follow Through</td>
</tr>
<tr>
<td>Prep.</td>
<td>Rqmts. Meeting</td>
<td>Follow Through</td>
</tr>
</tbody>
</table>

Subsystem Architecture Assessment Phase

<table>
<thead>
<tr>
<th>Subsystem 1</th>
<th>Subsystem 2</th>
<th>Subsystem N</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prep.</td>
<td>Arch. Meeting</td>
<td>Follow Through</td>
</tr>
<tr>
<td>Prep.</td>
<td>Arch. Meeting</td>
<td>Follow Through</td>
</tr>
<tr>
<td>Prep.</td>
<td>Arch. Meeting</td>
<td>Follow Through</td>
</tr>
</tbody>
</table>

System Architecture Assessment Summary Phase

<table>
<thead>
<tr>
<th>Prep.</th>
<th>Final Meeting</th>
<th>Follow Through</th>
</tr>
</thead>
</table>

Time (not to scale) →

© 2006 by Carnegie Mellon University Version 0.2
SRR Phase Overview

- Preparation Task
  - Architecture Assessment Procedure
  - Subsystem Requirements Review Preparatory Materials
  - Subsystem Requirements Review Checklist

- Meeting Task
  - Subsystem Requirements Review Presentation Materials
  - Subsystem Requirements Review Meeting Agenda
  - Subsystem Requirements Review Meeting Assessment Notes

- Follow-Through Task
  - Subsystem Requirements Review Meeting Outbrief
  - Updated Architecture Assessment Action Item List

Assessment Team
- Subsystem Requirements Team
- Subsystem Development Team
- Subsystem Architecture Team
SRR Preparation Task

Steps:

1. Subsystem Requirements Team provides access to the architecturally significant subsystem requirements as well as a summary of these requirements
2. Subsystem Architecture Team provides sample of planned Quality Cases
3. Subsystem Assessment Team reviews this information prior to the meeting
SRR Meeting Task

Steps:

1. Subsystem Requirements Team presents Summary of the architecturally significant subsystem requirements (organized by quality factor and quality subfactors)
2. Subsystem Assessment Team recommends Improvements
3. Subsystem Architecture Team presents sample of planned Quality Cases
4. Subsystem Assessment Team recommends Improvements
5. Assessment Team Manages Action Items
SRR Follow-Through Task

Steps:

1. Subsystem Assessment Team presents Outbrief
2. Subsystem Assessment Team develops and publishes Meeting Minutes containing recommendations for improving:
   - Architecturally significant subsystem requirements
   - Quality Cases
3. Assessment Team tailors and distributes updated Assessment Procedure and Assessment Training Material (for future requirements reviews)
4. Assessment Team captures Lessons Learned
5. Assessment Team Manages Action Items
SRR Roles and Responsibilities

The subsystem requirements review phase is performed by the following three teams:

- Subsystem Requirements Team
- Subsystem Architecture Team
- Subsystem Assessment Team
Subsystem Requirement Team

Responsibilities:
• Work with specialty engineering teams to engineer the architecturally significant subsystem requirements
• Provide these requirements to the subsystem architecture team in time to drive the subsystem architecture
• Provide the subsystem assessment team with access to these requirements sufficiently prior to the meeting
• Summarize these requirements at the requirements review meeting
• Answer questions from the assessment team (and architecture team)
Subsystem Architecture Team

Responsibilities:
• Develop a proposed representative sample of the architectural information to be presented during the coming subsystem architecture assessment meeting:
  - Architectural decisions and rationale
  - Supporting evidence
• Present this information to the subsystem assessment team
• Ask questions (if necessary) of the:
  - Subsystem requirements team (regarding architecturally significant requirements)
  - Subsystem assessment team (regarding the assessment process and adequacy of proposed sample architectural decisions, rationale, and evidence)
Subsystem Assessment Team

Responsibilities:
• Review supplied information prior to the requirements review meeting
• Ensure that the architecturally significant requirements are adequately engineered to support the subsystem architecture assessment.
• Ensure that the proposed architectural information to will be adequate to support the coming subsystem architecture assessment meeting
• Answer questions from and provide advice to the:
  - Requirements team regarding the architecturally significant requirements
  - Architecture team regarding what will be expected of them during the coming subsystem architecture assessment meeting
Subsystem Assessment Team

Responsibilities:

• Must include members having expertise in:
  - Requirements engineering and quality requirements
  - The system architecture quality assessment method
    (with all members having been trained in the method)

• Should include members having experience in the subsystem application domain(s) such as avionics, sensors, or weapons
SRR Discussion Questions

What is the two main objectives of the subsystem requirements review?
How often should subsystem requirements reviews be performed?
When should subsystem requirements reviews be performed?
What are the three tasks comprising the subsystem requirements review?
What are the objectives of these three tasks?
What teams are involved?
What are the responsibilities of these teams?
Subsystem Architecture Assessment (SAA)

System Architecture Assessment Initiation

- repeat for each subsystem being assessed

Subsystem Requirements Review → Subsystem Architecture Assessment

- done

- yes

- no

System Architecture Assessment Summary
SAA Topics

SAA Questions for the Attendees
SAA Phase Objectives
SAA Phase Principles
SAA Phase Challenges
SAA Phase Context
SAA Phase Overview:
  • SAA Preparation Task
  • SAA Meeting Task
  • SAA Follow-Through Task
SAA Roles and Responsibilities
Discussion
SAA Questions for the Attendees

As a *subsystem architect*, what are your biggest problems with respect to:

- Engineering the subsystem architecture?
- Ensuring that the subsystem architecture adequately meets its architecturally significant requirements?
- Internally reviewing/evaluating the quality of the subsystem architecture?
- Supporting independent assessments of the quality of your subsystem architecture?

As an *independent assessor* (e.g., PO of prime contractor, prime contractor of subcontractor), what are your biggest problems with respect to independently assessing the quality of an acquired subsystem’s architecture?
SAA Questions for the Attendees

Is the quality of your architectures being independently assessed?
How are your architectures being assessed?
Who is assessing your architectures?
What do you see as the biggest problems with respect to how your architectures are being assessed?
  • Are your assessors using an effective and efficient process for assessing your architectures?
  • Do you know what is expected of you during the system architecture assessments?
  • Do you develop adequate documentation as a natural part of the architecture process?
  • Is the architecture documentation you develop adequate to support assessments?
SAA Objectives

Assess Quality of Subsystem Architecture in terms of:

- Architectures support for its derived and allocated architecturally significant requirements
- Architectural Quality Cases
SAA Principles

Quality architecture assessments should be organized according to a quality model that defines quality factors (a.k.a., attributes, “ilities’) and their quality subfactors:

• Availability
• Interoperability
• Performance
  - Jitter, Response Time, Schedulability, and Throughput
• Portability
• Reliability
• Safety
• Security
• Usability
SAA Principles

The subsystem architects should know:
- What quality goals and requirements drove the development of their architectures.
- What architectural decisions they made.
- Why they made these decisions.
- Where these decisions are documented.

Because the subsystem architects should already have documented this information as a natural part of their architecting method, little new documentation should be necessary for the subsystem architects to make their cases to the subsystem assessment team.

The subsystem architects are responsible for making their own cases that their architectures adequately support their derived and allocated quality requirements.
SAA Phase Challenges

Architects may not have developed quality cases as a natural part of their architecting process:
• Architectural documentation typically not organized by quality factors.
• Quality case evidence is often buried in and scattered throughout massive amounts of architectural documentation.
• Architectural models (e.g., UML) often do not address support for quality requirements.

Architecture assessments may not be:
• Mandated by contract or development process
• Scheduled and funded

Managers feel schedule pressures do not allow time for assessment.
### SAA Context

<table>
<thead>
<tr>
<th>System Architecture Assessment Initiation Phase</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prep.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Subsystem Architecture Assessment Phase</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subsystem Requirements Review Phase</td>
</tr>
<tr>
<td>Prep.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Subsystem Architecture Assessment Phase</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prep.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Subsystem N Architecture Assessment Phase</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subsystem Requirements Review Phase</td>
</tr>
<tr>
<td>Prep.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Subsystem Architecture Assessment Phase</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prep.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>System Architecture Assessment Summary Phase</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prep.</td>
</tr>
</tbody>
</table>

---

**Time (not to scale)**
SAA Preparation Task

Steps:
1. Subsystem Assessment Team Provides Assessment Checklist
2. Subsystem Architecture Team Gathers (Generates) and Makes Available *Preparatory* Materials:
   - Subsystem Architecture Overview
   - Updated Quality Requirements
   - Quality Cases including Arguments and Evidence
3. Subsystem Architecture Team Gathers (Generates) and Makes Available *Presentation* Materials
4. Subsystem Assessment Team:
   - Reads Materials
   - Generates RFIs and RFAs
5. Teams Collaborate to Organize Assessment Meeting (Attendees, Time, Location, Agenda, Invitation)
SAA Meeting Task

Steps:

1. Subsystem Architecture Team:
   • Introduces Subsystem Architecture
     (purpose, location, context, functions)
   • Reviews Architecturally-Significant Requirements
   • Introduces Subsystem Architecture
     (components, relationships, major decisions, trade-offs)
   • Present Quality Cases
     (claims, arguments, and evidence)

2. Subsystem Assessment Team:
   • Probes Architecture (quality case by quality case)
   • Manages Action Items
SAA Follow-Through Task

Steps:

1. Subsystem Assessment Team:
   • Develops Consensus
   • Produces, Reviews, and Presents Meeting Outbrief
   • Produces, Reviews, and Presents Subsystem Assessment Report
   • Manages Action Items
   • Captures Lessons Learned
   • Updates Assessment Method and Training Materials
SAA Roles and Responsibilities

The Subsystem Architecture Assessment Phase is performed by the following teams:

- Subsystem Architecture Team
- Subsystem Assessment Team
Subsystem Architecture Team

Responsibilities:

• Develop the architectural information to be presented during the meeting:
  - Architectural decisions and rationale
  - Supporting evidence
• Present this information to the subsystem assessment team
• Answer probing questions raised by the subsystem assessment team:
Subsystem Assessment Team

Responsibilities:

- Review supplied information prior to the subsystem architecture assessment meeting
- Assess the quality of the subsystem architecture:
  - Actively listen to the quality cases presented by the subsystem architecture team
  - Ask probing questions of Architects
SAA Discussion

Should the quality cases be developed as a:

- Natural part of the architecting process?
- Part of the assessment process?

How does the answer to the previous question affect the amount of time needed to prepare for the assessment meeting?

Which team has the most work to do during each task?

How should the development of the subsystem assessment report be divided up between members of the assessment team?
System Architecture Assessment Summary (SAAS)

1. System Architecture Assessment Initiation
2. Repeat for each subsystem being assessed:
   - Subsystem Requirements Review
   - Subsystem Architecture Assessment
3. If no, go back to System Architecture Assessment Initiation.
4. If yes, go to System Architecture Assessment Summary.
SAAS Topics

SAAS Questions for the Attendees
SAAS Phase Objectives
SAAS Phase Principles
SAAS Phase Challenges
SAAS Phase Context
SAAS Phase Overview:
  • SAAS Preparation Task
  • SAAS Meeting Task
  • SAAS Follow-Through Task
SAAS Roles and Responsibilities
Discussion
SAAS Questions for the Attendees

How do you summarize the results of subsystem assessments at the system level?

Should the system architecture assessment summary phase be performed:
  • Once at the end?
  • On an ongoing rolling-wave basis?
SAAS Objectives

Collect previous Subsystem Architecture Assessment Results
Create System Architecture Assessment Summarize Results
Capture Method Lessons Learned
Update Assessment Method and Training Materials
SAAS Principles

All subsystems are not equally important.
All quality factors are not equally important for different subsystems.
It is probably better to concentrate on identifying problem/risk areas so that they can be fixed than to provide an overall summary assessment result.
SAAS Phase Challenges

How should subsystem findings be summarized without ending up comparing apples and oranges?

- Average Subsystem Architecture Quality
- Worst Subsystem Architecture Quality
- Union of Subsystem Architecture Qualities

Executive management may demand simplistic single number summary of system architecture.
SAAS Context

System Architecture Assessment Initiation Phase

<table>
<thead>
<tr>
<th></th>
<th>Prep.</th>
<th>Initial Meeting</th>
<th>Follow Through</th>
</tr>
</thead>
</table>

Subsystem 1 Architecture Assessment

<table>
<thead>
<tr>
<th>Subsystem Requirements Review Phase</th>
<th>Subsystem Architecture Assessment Phase</th>
</tr>
</thead>
</table>

Subsystem 2 Architecture Assessment

<table>
<thead>
<tr>
<th>Subsystem Requirements Review Phase</th>
<th>Subsystem Architecture Assessment Phase</th>
</tr>
</thead>
</table>

---

Subsystem N Architecture Assessment

<table>
<thead>
<tr>
<th>Subsystem Requirements Review Phase</th>
<th>Subsystem Architecture Assessment Phase</th>
</tr>
</thead>
</table>

System Architecture Assessment Summary Phase

<table>
<thead>
<tr>
<th></th>
<th>Prep.</th>
<th>Final Meeting</th>
<th>Follow Through</th>
</tr>
</thead>
</table>

Time (not to scale) →

© 2006 by Carnegie Mellon University

Version 0.2

QUASAR Method. - page 90
SAAS Phase Overview

- Preparation Task
  - System Summary Subsystem Matrix
  - System Summary Meeting Presentation Materials
  - System Architecture Assessment Summary Meeting Agenda
  - System Architecture Assessment Meeting Assessor Notes

- Meeting Task
  - Subsystem Architecture Assessment Summary Meeting Report
  - System Architecture Quality Assessment Summary Report
  - Updated Architecture Assessment Action Item List

- Follow-Through Task
  - Architecture Assessment Training Materials
  - Architecture Assessment Procedure

- System Assessment Team
  - System Architecture Team

© 2006 by Carnegie Mellon University

Version 0.2

QUASAR Method. - page 91
SAAS Preparation Task

Steps:

1. System Assessment Team:
   • Collects Subsystem Architecture Assessment Results
   • Summarizes Subsystem Architecture Assessment Results
     • Develops Subsystem Architecture Support Matrix
   • Identifies Primary Stakeholders
   • Produces, Reviews, and Distributes:
     • System Architecture Quality Assessment Summary Report
     • Preparatory Materials
     • Meeting Agenda
   • Organizes Meeting
SAAS Meeting Task

Steps:

1. System Assessment Team:
   • Restates Assessment Objectives
   • Summarizes Assessment Method
   • Summarizes Quality of Subsystem Architectures
   • Summarizes Quality of System Architecture
   • Solicits Feedback
   • Captures Lessons Learned

2. System Architecture Team:
   • Captures Lessons Learned
SAAS Follow-Through Task

Steps:

1. System Assessment Team:
   • Updates and Distributes the System Architecture Assessment Summary Report
   • Manages Action Items
   • Updates Assessment Method and Training Materials

2. System Architecture Team:
   • Updates Architecture Method and Training Materials
SAAS Responsibilities

System Assessment Team:
• Develop and Present System-Level Architecture Assessment Summary Results
• Capture Lessons Learned
• Update Assessment Method and Training Materials

System Architecture Team:
• Validate Assessment Results
• Capture Lessons Learned
• Update Architecture Method and Training Materials

Management Team:
• Manage Architectural Risks
SAAS Discussion

For a given quality factor, what is the best way to summarize the quality of the system architecture in terms of the quality of the architecture of the main subsystems?

- Average subsystem quality?
- Worst subsystem quality?
- Keep separate by listing individually?

What is the best way to summarize across all quality factors?

- Average value?
- Worst value?
- Keep separate by listing individually?
QUASAR Today and Tomorrow

Today:
- In-use on massive DoD Program
- Handbook published
- Provided as SEI Service

Future Plans:
- More Conference Tutorials
- QUASAR Training Materials and Classes
- QUASAR Articles
- Use and Validation on more Programs
- QUASAR Book
QUASAR Handbook

Intended Audiences:
• Acquisition Personnel
• Developers (Architects and Requirements Engineers)
• Subject Matter Experts (domain, specialty engineering)
• Consultants
• Trainers

Objectives:
• Completely Document the QUASAR method
• Enable Readers to start using QUASAR

Description:
• Very Complete
• Too comprehensive to be good first introduction
Questions?

For more information, contact:
Donald Firesmith
Acquisition Support Program
Software Engineering Institute
dgf@sei.cmu.edu
9th Annual System Engineering Conference
October 23, 2006

National Defense Industry Association (NDIA)
System Engineering Division

Systems Level Configuration Management

Al Florence

The MITRE Corporation

The author’s affiliation with The MITRE Corporation is provided for identification purposes only, and is not intended to convey or imply MITRE’s concurrence with, or support for, the positions, opinions or viewpoints expressed by this author.
Contents of Course

- Introduction
  - Configuration Management Concepts
  - Configuration Management in Detail
  - Tailoring Configuration Management
  - Points to Remember
  - References / Suggested Reading
  - Questions / Answers / Discussion
  - Contact Information
Introduction

Course Objectives

Provide students with an understanding of:

- Configuration Management (CM)
- Importance of CM
- Identification of Configuration Items (CIs)
- Baselines
- Controlling Changes to CIs
- Configuration Control Boards
- Classes of Changes
- Conducting Impact Assessments on Requested Changes
- Configuration Status Accounting
- Configuration Management Audits
- CM Responsibilities of Stakeholders
- CM Relationships between Acquirer and Supplier
Introduction (continued)

Why CM?

- CM ensures that the current configuration of items are known throughout their lifecycle.
- CM ensures that changes to the configuration of evolving items are correct, controlled, managed, and documented.
Introduction (continued)

What is CM?

◆ CM is a discipline applying technical and administrative direction and surveillance to:
  – Identifying and documenting the physical, functional, and performance characteristics of items
  – Baselineing those characteristics
  – Controlling changes to those characteristic
  – Providing status on those characteristics
  – Conducting audits on those characteristics

◆ The CM tasks that produce these results are:
  – Configuration Identification
  – Configuration Control
  – Configuration Status Accounting
  – Configuration Management Audits
Application of CM

The CM concepts presented in this tutorial can be applied at the enterprise/systems/subsystem/program/project level to:

- Hardware
- Software
- Facilities
Introduction (continued)

**Capability Maturity Model Integration (CMMI®)**

The Software Engineering Institute’s CMMI® has a supporting Process Area that requires organizations that are developing systems to conduct a minimum set of CM tasks on the development and maintenance of products in order to achieve CMMI compliance.
CMMI (Continued)

Configuration Management Process Area (continued)

CMMI - Configuration Management

- SG 1 Establish Baselines
  - SP 1.1 Identify Configuration Items
  - SP 1.2 Establish a Configuration Management System
  - SP 1.3 Create or Release Baselines

- SG 2 Track and Control Changes
  - SP 2.1 Track Change Requests
  - SP 2.2 Control Configuration Items

- SG 3 Establish Integrity
  - SP 3.1 Establish Configuration Management Records
  - SP 3.2 Perform Configuration Audits

SG – Specific Goal
SP – Specific Practice
CMMI (completed)

Configuration Management Process Area (completed)

CMMI - Configuration Management

◆ GG 2 Institutionalize a Managed Process
  – GP 2.1 Establish an Organizational Policy
  – GP 2.2 Plan the Process
  – GP 2.3 Provide Resources
  – GP 2.4 Assign Responsibility
  – GP 2.5 Train People
  – GP 2.6 Manage Configurations
  – GP 2.7 Identify and Involve Relevant Stakeholders
  – GP 2.8 Monitor and Control the Process
  – GP 2.9 Objectively Evaluate Adherence
  – GP 2.10 Review Status with Higher Level Management

GG – Generic Goal
GP – Generic Practice

AI Florence MITRE
Introduction (continued)

Some Levels of CM

Enterprise CM

Supplier CM

Development CM
- Formal CM
- CI Characteristics
  - Physical
  - Function
  - Performance

Internal CM
- Design
- Implementation
- Code
- Test
- Process Documentation

Acquirer CM

Development
- Formal CM
- CI Characteristics
  - Physical
  - Function
  - Performance

Internal CM
- Business Cases
- Business Practices
- Budgets

*Control Changes:
  - Cost
  - Schedule
  - Interfaces

*Could be a different contractor

AI Florence MITRE
Introduction (continued)

Some Levels of CM (concluded)

◆ **Enterprise CM**
  – Covers all CM required for the entire enterprise (Acquirer/Supplier)

◆ **Supplier CM - Formal CM**
  – Development CM that concerns high level contractual issues such as specifications *(What shall be accomplished?)*

◆ **Supplier CM - Internal CM**
  – Development CM that concerns lower level contractual issues such as design, implementation, test, plans *(How is it accomplished?)*

◆ **Acquirer CM - Development Formal CM**
  – Acquirer CM that concerns high level contractual issues such as specifications *(What shall be accomplished?)*

◆ **Acquirer CM - Internal CM**
  – Acquirer CM that concerns internal business issues

◆ **Operational & Maintenance CM**
  – CM conducted after the system has been delivered and in operation
This focus is chosen to serve as an example that can be applied, as appropriate, to other levels of CM and because this is one area in development that gets projects in trouble very quickly if not done properly.
Where are we?

- Introduction
- Configuration Management Concepts
  - Configuration Management in Detail
  - Tailoring Configuration Management
  - Points to Remember
  - References / Suggested Reading
  - Questions / Answers / Discussion
  - Contact Information
CM Concepts

System

Configuration Identification

Configuration Item

Configuration Control Board

Baseline

Configuration Control

Technical Review Board

Configuration Management Audits – Configuration Status Accounting
CM Concepts (continued)

System
◆ A composite of items (e.g., hardware, software, facilities, personnel, material, services, and techniques) required to perform a complete operational role

Configuration Identification
◆ The identified configuration of items such as hardware, software, and facilities within a system, and their physical, functional, and performance characteristics

Configuration Item
◆ An identified configuration of an item, or a portion of its parts, that is designated for change control
CM Concepts (continued)

**Configuration Item**

Represents the characteristics of a Configuration Item

- **Functional and performance characteristics**
  - Rolls down hill at 10 mph

- **Physical characteristic**
CM Concepts (continued)

**Baseline**

- The approved and fixed (baselined) configuration of a CI at a specific time in its lifecycle that serves as a reference point for change control
  - CIs are used for visibility
  - Baselines are used for control

![Diagram showing CI and Baselined CI with Visibility and Control labels]
CM Concepts (continued)

**Configuration Control**

◆ The systematic
  – evaluation
  – coordination
  – approval or disapproval, and
  – implementation

of changes to the physical, functional, and performance characteristics of a baselined CI
CM Concepts (continued)

**Configuration Control Board (CCB)**

- Establishes baselines for CIs
- Reviews and approves / disapproves / defers Change Requests to CIs
- Membership comprised of management and other stakeholders and supported by subject matter experts (SMEs)
  - Project Management
  - Systems Engineering
  - Software/Hardware Engineering
  - Test Engineering
  - Quality Assurance
  - Configuration Management
- Chaired by the program / project manager or designee
CM Concepts (continued)

Technical Review Board (TRB)

- Provides technical and programmatic support to the CCB
  - Conducts impact assessment on change requests (CRs) to baselined CIs
  - Makes approval / disapproval recommendations to the CCB

- Membership comprised of program / project personnel and subject matter experts

- Chaired by a technical manager
CM Concepts (concluded)

Configuration Management Audits

- Audits are conducted on CM tasks by the CM organization and Quality Assurance to ensure that CM is being executed as described in CM process documentation.

- At the end of development and prior to delivery, audits are conducted for the Acquirer to:
  - Ensure that all products comply with their requirements.
  - Ensure that all products comply with their design documents such as the software design, hardware design and facilities design documents.
CM Concepts (continued)

Configuration Status Accounting (CSA)

- CSA is performed to gather, correlate, maintain and provide status on controlled products (CIs), and on CM tasks.
Where are we?

- Introduction
- Configuration Management Concepts
- Configuration Management in Detail
- Tailoring Configuration Management
- Points to Remember
- References / Suggested Reading
- Questions / Answers / Discussion
- Contact Information
Configuration Management in Detail

This section will cover the following:

- Configuration Management Planning
- Configuration Identification
- Configuration Control
- Configuration Status Accounting
- Configuration Management Audits
CM Planning

- CM planning is essential if CM is to be effectively applied throughout the lifecycle of products.

- When conducting CM planning for a particular program/project, the size, type, and scope of the applications and the program/project needs to be accounted for in order to provide the correct amount of CM.

- Big Project requires Big CM.

- Little Project requires Little CM.
CM Planning (continued)

Planning Activities Are:
- Identifying CM Tasks
- Defining CM Roles and Responsibilities
- Selecting CM Tools
- Determining CM Resources
- Determining CM Training
- Defining CM Metrics
- Developing the CM Plan which is the output of planning
CM Planning (continued)

Identifying CM Tasks

CM Planning

- Configuration Identification
- Configuration Control
- Configuration Audits
- Configuration Status Accounting
CM Planning (continued)

Defining CM Roles and Responsibilities

- CM is NOT solely the responsibility of the CM organization
- CM involves all stakeholders of the program / project
- All organizations involved with the engineering and development of program / projects products have CM roles and responsibilities
CM Planning (continued)
Defining CM Roles and Responsibilities (continued)

Configuration Management Manager

- Primary responsibility for the development of the CM plan
- Responsible for overseeing tasks assigned to the CM Organization and ensuring that they are performed
- Submits CM plans for approval
- Serves on the CCB (provides scribe)
- Conducts impact assessments on CRs
CM Planning (continued)
Defining CM Roles and Responsibilities (continued)

Configuration Management Practitioners

- Primary responsibility for developing, implementing and maintaining CM plans, processes, procedures, and tools
- Responsible for CM Repository*
- Prepare agendas and minutes for CCB meetings
- Administration of CRs
- Tracks the implementation of approved CRs
- Conducts CM Process audits
- Responsible for CM Status Accounting

*CM Repository stores CM documentation, CM records, CM artifacts, etc.
CM Planning (continued)
Defining CM Roles and Responsibilities (continued)

◆ **Program/Project Manager**
  – Authorizes CM plans
  – Ensures adequate CM resources are provided
  – Enforces CM tasks
  – Chairs the Configuration Control Board (CCB)

◆ **Quality Assurance**
  – Audits CM activities to ensure that CM tasks are conducted in accordance with documented plans and processes
  – Conducts impact assessments on CRs
  – Serves on the CCB
CM Planning (continued)
Defining CM Roles and Responsibilities (continued)

◆ **Systems Engineering**
  - Primary responsibility for definition and identification of system level CIs
  - Conducts impact assessments on CRs
  - Conducts product audits at end of development
  - Serves on the CCB

◆ **Hardware / Software Engineering**
  - Primary responsibility for definition, identification and implementation of hardware / software CIs
  - Conducts impact assessments on CRs
  - Conducts product audits at end of development
  - Serves on the CCB
CM Planning (continued)
Defining CM Roles and Responsibilities (concluded)

◆ Test Engineering
  – Responsible for testing CIs
  – Conducts impact assessments on CRs
  – Provide test-related CM artifacts for the CM repository
  – Conducts product audits at end of development
  – Serves on the CCB
### CM Planning (continued)
#### Defining CM Roles and Responsibilities (concluded)

<table>
<thead>
<tr>
<th>Organizations</th>
<th>CM Planning</th>
<th>Configuration Identification</th>
<th>Configuration Control</th>
<th>Configuration Status Accounting</th>
<th>Configuration Management Audits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project Manager</td>
<td>Approves</td>
<td>Supports</td>
<td>Approves Changes</td>
<td>Receives Reports</td>
<td>Supports</td>
</tr>
<tr>
<td>Configuration Management Organization</td>
<td>Conducts</td>
<td>Facilitates</td>
<td>Facilitates process</td>
<td>Conducts</td>
<td>Facilitates Conducts</td>
</tr>
<tr>
<td>Quality Assurance</td>
<td>Audits Process Artifacts</td>
<td>Audits Process Artifacts</td>
<td>Audits Process Artifacts</td>
<td>Receives &amp; Audits Reports</td>
<td>Witnesses Conducts</td>
</tr>
<tr>
<td>Systems Engineering</td>
<td>Supports</td>
<td>Conducts</td>
<td>Supports</td>
<td>Receives Reports</td>
<td>Conducts Product Audits</td>
</tr>
<tr>
<td>HW/SW Engineering</td>
<td>Supports</td>
<td>Conducts</td>
<td>Supports</td>
<td>Receives Reports</td>
<td>Conducts Product Audits</td>
</tr>
<tr>
<td>Test Engineering</td>
<td>Supports</td>
<td>Supports</td>
<td>Supports</td>
<td>Receives Reports</td>
<td>Conducts Product Audits</td>
</tr>
</tbody>
</table>

* CM facilitates Product Audits and conducts other CM audits
** QA may also facilitate Product Audits and conduct other CM audits
Support involves providing information and subject matter expertise, and reviewing artifacts

Al Florence  MITRE
Selecting CM Tools

CM tools should be used to:
- Store CM documentation and artifacts
- Control versions
- Track and status CRs
- Support administration and communication
  - Create documents and reports
  - Develop presentations
  - Produce schedules
  - Collect measurements
  - Conduct analysis / create metrics
CM Planning (continued)

Determining CM Resources

CM resources are comprised of:

- CM personnel
- Facilities
- Funding
- Equipment
- Tools
- Supplies
- Administrative support
CM Planning (continued)

Determining CM Training

CM training may include:

- Project management level CM orientation
  - CM roles and responsibilities

- Program / project staff CM orientation and training
  - Orientation on CM roles and responsibilities
  - Training on CM activities

- CM practitioner training
  - CM roles and responsibilities
  - In-depth CM training on CM activities
  - Use of CM tools and CM repository
CM Planning (continued)

Determining CM Metrics

Measurements are collected during the execution of the CM activities. The following are some typical CM measurements:

◆ Baselines
  – Number of items pending baselining
  – Number of actual items baselined

◆ Change Requests
  – Number of CRs approved
  – Number of CRs implemented

◆ For each CR
  – Date approved
  – Planned date closed
  – Actual date closed
CM Planning (continued)

Determining CM Metrics

- The measurements analyzed which result in metrics which can be shown in charts, graphs, tables, etc.

- **Metrics can be used to:**
  - Provide status to management on the CM activities, products and services
  - Determine behavior of the CM process used to conduct the activities that produce products and services
  - Make management decisions and corrections to bring products and activities under control when required
  - Identify areas where the CM or engineering process is unstable, which may lead to process improvement
## Change Requests as of November 4, 2003

<table>
<thead>
<tr>
<th>Number Approved</th>
<th>Number Implemented</th>
<th>Number Not Implemented</th>
<th>Number Missed Due Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>35</td>
<td>20</td>
<td>10</td>
<td>4</td>
</tr>
</tbody>
</table>

![Bar Chart](image_url)

**Change Requests**
- Approved: 35
- Implemented: 20
- Not Implemented: 10
- Missed Due Date: 4
CM Planning (continued)

Developing the CM Plan

◆ The purpose of the CM Plan is to describe the processes, products, and organizational responsibilities required to implement effective CM functions on programs / projects

◆ The Plan presents CM as it is applied to a particular program / project and tailors the CM approach appropriately to the scope of the application
  - A large application needs BIG CM
  - A small application needs small CM

◆ More on tailoring CM will be presented later

The CM Plan should **not** include detailed CM processes that require frequent update
The Plan contains the following:

- **Introduction of the plan’s purpose, scope, and contents**
- **An overview of the CM tasks**
  - Configuration Identification
  - Configuration Control
  - Configuration Audits
  - Configuration Status Accounting
- **Organizational CM roles and responsibilities**
- **CM risk management**
- **CM resources**
- **CM metrics**
- **High level milestone and schedules relating to CM**
Configuration Management in Detail

- Configuration Management Planning
- Configuration Identification
  - Configuration Items
  - Baselines
- Configuration Control
- Configuration Status Accounting
- Configuration Management Audits
Configuration Identification

- Configuration Identification is established in the form of documentation of items that becomes more detailed as development proceeds.
- It is important to assign unique identifiers to items.
  - Supports version identification and control.
Three levels of Configuration Identification are established:

- **Functional Configuration Identification (FCI)**
- **Allocated Configuration Identification (ACI)**
- **Physical Configuration Identification (PCI)**

<table>
<thead>
<tr>
<th>Lifecycle Phases</th>
<th>Design</th>
<th>Implementation</th>
<th>Test</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>FCI</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ACI</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PCI</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Conceptual System Requirements

Hardware Software Facilities Requirements

AI Florence MITRE
Functional Configuration Identification (FCI)\textsuperscript{1, 2}

The identified system and system items and their physical, functional, and performance characteristics which are documented in a System CI Specification.
Allocated Configuration Identification (ACI) —\(^1, 2\)

Later in development the physical, functional, and performance characteristics of the system are allocated to lower level entities: software, hardware, facilities, and documented as CI Specifications.
Physical Configuration Identification (PCI)\textsuperscript{1, 2}

Finally, the products of the developed system: software, hardware, facilities are defined in a series of Product CI Specifications that describe the as-built system.

As-built System

Product CI Specifications
Configuration Identification (concluded)

- **Functional Configuration Identification (FCI)**
  - System CI Specification
  - Conceptual Phase

- **Allocated Configuration Identification (ACI)**
  - Software CI Specifications
  - Hardware CI Specifications
  - Facilities CI Specifications
  - Development Phases

- **Physical Configuration Identification (PCI)**
  - Designed/Built/Tested Entities
  - Design/Built/Tested Entities
  - Design/Built/Tested Entities
  - Operational Phase

- **Product CI Specifications**

- **As-Built Products**
CM Concepts in Detail (continued)

Configuration Items

**Configuration Identification and Configuration Items**

- Configuration Identification is an activity that identifies items and their characteristics: physical, functional, and performance.
- Not all items that are identified need be controlled at the same level of rigor.
- Configuration Items are selected for **formal change control** from items identified.

<table>
<thead>
<tr>
<th>Configuration Identification - Software</th>
</tr>
</thead>
<tbody>
<tr>
<td>*Operating System</td>
</tr>
</tbody>
</table>

*These are commercial products not subject to change – In operation (production) everything is under CM control.

**Applications software in development that is subject to change.
Baselines

Baselines are established at strategic points in a system lifecycle. Three baselines may be defined:

- Functional Baseline (FBL)
- Allocated Baseline (ABL)
- Product Baseline (PBL)
Functional Baseline (FBL)—Established for the CI of the System Functional Configuration Identification when System Requirements Phase of the system lifecycle is completed.  

- System CI Specification is under formal change control by the CCB.  
- Time is marked by the end of the Systems Requirements Phase.
Baselines (continued)

Allocated Baseline

Allocated Baseline (ABL)—¹, ² Established for the CIs of the Allocated Configuration Identification later in development for software, hardware, facilities, etc.

Software CI Specification

Hardware CI Specification

Facilities CI Specification

End of Software/Hardware/Facilities Requirements Phase

Under Formal Change Control by the CCB

Time

ABL
Baselines (continued)

**Product Baseline**

**Product Baseline (PBL)**—1, 2 Established for the developed products (CIs) of the Physical Configuration Identification at the end of development.

Under Formal Change Control by the CCB During Operation

End of Development Prior to Delivery to Customer

Al Florence MITRE
Baselines (continued)

- **Functional Baselines (FBL)** for:
  - System CI

- **Allocated Baselines (ABL)** for:
  - Development Phases
    - Hardware CIs
    - Software CIs
    - Facilities CIs

- **Product Baselines (PBL)** for:
  - Operational Phase
    - Hardware CIs
    - Software CIs
    - Facilities CIs

As-Built Products
It gets more complex:

- As development progresses CIs evolve and include more detail:
  - Initially the CIs are represented as requirements documented in CI Requirements Specifications
  - Later the CIs are represented in:
    - Design documents
    - Test plans
    - Code (for software)
    - Test procedures
    - Test results
  - During development only CIs that have achieved the Functional Baseline and the Allocated Baseline for the CI Specifications are designated for formal CCB control*

* As described for this presentation and as reflected in references 1 and 2.
Example

A NASA spacecraft (Galileo) orbits Jupiter and releases a probe to conduct scientific experiments on the planet’s atmosphere.
Functional Configuration Identification

1. Identify 3 configuration Items of one subsystem – earth station, spacecraft, probe
2. List the some physical, functional, and performance characteristics of one item
3. Allocate these characteristics to software, hardware, systems, and facilities as appropriate
4. Write these allocations as “shall” requirements
Example (continued)

◆ **Jupiter**
  - A gas planet
  - No terrestrial surface
  - High atmospheric pressures
  - Compresses to a hard state at surface

◆ **Spacecraft**
  - 45 minute one way communication delay at speed of light to/from earth/spacecraft

◆ **Probe**
  - Autonomous control
  - One way communications from probe to spacecraft
  - 40 minute science data collection life
  - Implodes due to high atmospheric pressures
Example (continued)

**PROBE**

**Configuration Items:**
1. Parachute
2. Heat Shield
3. Science Instruments

**Heat Shield characteristics:**

**A. Physical:**
A.1 Three feet in diameter
A.2 Half inch thick graphite
A.3 Perfect circular and spherical curvature shape

**B. Functional:**
B.1 Protect probe from over heating

**C. Performance:**
C.1 Jettisoned 48 hours 20 minutes after separation from spacecraft
C.2 Jettisoned within one second of jettison command
C.3 Withstands temperatures up to 500 degrees Fahrenheit
Example (continued)

Probe Heat Shield

Heat Shield Allocations:

A. Physical:
   A.1 Three feet in diameter
   A.2 Half inch thick graphite
   A.3 Perfect circular and spherical curvature shape

B. Functional
   B.1 Protect probe from over heating

C. Performance
   C.1 Jettisoned 48 hours 20 minutes after separation from spacecraft
   C.2 Jettisoned within one second of jettison command
   C.3 Withstands temperatures up to 500 degrees Fahrenheit

Hardware (HW)
System
Software/HW
Hardware
Example (completed)

Probe Heat Shield

Allocated Shall Requirements:

HW1 Heat Shield physical hardware requirements
  HW1.1 The heat shield shall be three feet in diameter.
  HW1.2 The heat shield shall be constructed of half inch thick graphite.
  HW1.3 The heat shield shall be of a perfect circular and spherical curvature shape.

Sys1 Heat Shield functional system requirements
  Sys1.1 The heat shield shall protect the probe from over heating.

SW1 Heat Shield software performance requirements (Also Hardware)
  SW1.1 The heat shield shall be jettisoned 48 hours 20 minutes after separation from spacecraft.
  SW1.2 The heat shield shall be jettisoned within one second of jettison command.

HW2 Heat Shield hardware performance requirements
  HW2.1 The heat shield shall withstand temperatures up to 500 degrees Fahrenheit.

Al Florence MITRE
Configuration Management in Detail

- Configuration Management Planning
- Configuration Identification

- Configuration Control
  - CCB and TRB

- Configuration Management Audits
- Configuration Status Accounting
Configuration Control

Configuration Item

Physical characteristic

Functional and performance characteristic

Constraint

Gravel

3% Grade

Change Request

Rolls down hill at 10 mph

Less than 3 mph wind

Need to control the configuration of physical, functional, and performance characteristic

Slides down hill at 15 mph

If not we might get something really dumb or suffer a catastrophic failure

Al Florence MITRE
How are Changes Accomplished?

- **Request Change**
  - Someone requests a change to a CI using a CR form

- **Evaluate Change**
  - The TRB conducts an impact assessment to ensure that all stakeholders evaluate the impact against their interests

- **Approve Change**
  - The CCB approves, disapproves or defers the CR

- **Implement Change**
  - The change is implemented in all affected items

- **Track Changes**
  - Changes are audited to verify that they are implemented as approved and tracked against the change schedule
CM Concepts (concluded)

CR Example

<table>
<thead>
<tr>
<th>CR #</th>
<th>Date: 12/4/2003</th>
<th>Requestor: ET</th>
<th>Class:</th>
<th>II</th>
</tr>
</thead>
</table>

**Problem:** A requirement to deploy the probe’s parachute does not exist

**Change:** Add the following requirement: The probe’s parachute shall be deployed 10 seconds after the heat shield has been jettisoned

**Impacts:** Enter figures for cost and schedule and list affected interfaces or “None” and attach impact assessments

Systems:
Hardware:
Software:
Test:
Configuration Management:
Quality Assurance:
Contracts:
Other [Specify]:

**Approve:**
TRB Date: Chair:
CCB Date: Chair:

**Disapprove:**
TRB Date: Chair:
CCB Date: Chair:

**Assignee:**
Due Date:
Configuration Control (continued)

Change Flow

1. Request Change
   - Supplier or Acquirer

2. Evaluate Change
   - TRB

3. Approve Change
   - CCB

4. Implement Change
   - Owner of item

5. Track Change
   - CM staff and owner of item

AI Florence MITRE
Impact Assessments

- Impact assessments need to be conducted by all stakeholders:
  - Systems
  - Hardware
  - Software
  - Test
  - Configuration Management
  - Quality Assurance
  - Contracts
  - Others

- On CI characteristics:
  - Physical
  - Functional
  - Performance

- Against their interests:
  - Cost
  - Schedule
  - Interface
Classification of Changes

At least two types of changes can be defined:

1, 2 Class I—affects the Acquirer’s interest in one or more of these factors:
- Physical characteristics
- Functional capability
- Performance
- External interfaces
- Cost
- Schedule

Supplier must submit change to the Acquirer for approval before implementation (Based on Thresholds)
1, 2 Class II

- Does not affect any of the class I factors
- Affects changes such as:
  - Spelling or typographical errors
  - Addition of clarifying comments
  - Changes that do not affect external interfaces, change functionality or degrade performance

Supplier may implement it without Acquirer’s approval but must inform Acquirer of change
CM Concepts in Detail (continued)

CCB and TRB

- CCB is a formal board dealing with contractual items such as requirements specifications
- CCB membership consists of senior and program management
  - Very busy, and “$$” to deal with lower-level items
- TRBs are less formal and deal with internal control of items such as design, implementation, and test
- TRBs act as a winnowing agent on items that should not go to the CCB
- TRBs conduct impact assessments on CRs and make recommendation to the CCB of approval or rejection
- TRB membership consist of program technical management and subject matter experts, “$$” that provides technical support to the CCB
CCB and TRB (continued)

CCB and TRB Hierarchy

- Acquirer CCB
  - TRB
  - Supplier Program CCB
    - TRB
    - Supplier Project A CCB
      - TRB
      - Supplier Project N CCB
        - TRB

Acquirer (Customer)
Supplier (Contractor)
CCB and TRB Change Flow

Change Request

- Project TRB
  - Class I
  - Incorporate

- Class II
  - Can reject
    - Affects Higher CI
      - Program TRB
        - Incorporate
        - Affects Higher CI
          - Acquirer TRB
            - Incorporate

Next higher level TRB needs to verify the assessments and the recommendation

- Impact Assessment & CCB Recommendation
  - Project CCB
    - Affects Higher CI
      - Yes
        - Approve
      - No
        - Incorporate

- Next Slide
  - Can reject

- Done
  - No
  - Yes

AI Florence MITRE
CCB and TRB (concluded)

CCB and TRB Change Flow (concluded)

Prior Slide -> Program CCB

- Affects Higher CI
  - Yes: Acquirer CCB
  - No: Can reject

Can reject -> Approve

- Incorporate
  - Project CCB
    - Incorporate

Approve -> Incorporate

- Program CCB
  - Incorporate

Can reject -> Incorporate

Based on thresholds

AI Florence MITRE
Configuration Management in Detail

- Configuration Identification
- Configuration Items
- Baselines
- Configuration Control
- Configuration Management Audits
- Configuration Status Accounting
CM Audits

- Functional Configuration Audits (FCA) and Physical Configuration Audits (PCA) are conducted by Engineering and facilitated by CM and/or QA.

- Other audits conducted by QA and CM may include:
  - Audits of CM Repository that contains CM records, documentation, processes, procedures, artifacts, etc.
  - Audits of Program/Project organizations to ensure CM process is being followed.
  - Audits of status of approved CRs.
  - Audits to ensure that CIs are consistent with CM records.
CM Audits (continued)

**Functional Configuration Audit (FCA)**

- A formal examination of test results of the as-built functional configuration of CIs, prior to acceptance, to verify that the CIs have satisfied their specified requirements \(^1, 2\)

- This audit is conducted by the Supplier for the Acquirer and attended by
  - Management
  - System Engineering
  - Hardware / Software Engineering
  - Test Engineering
  - QA and CM
  - Contracts

of both the Acquirer and Supplier
CM Audits (continued)

FCA (completed)

Functional

- Requirements Specifications
- Requirements Traceability
- Test Plans
- Test Scenarios

Testing

- Products
- Tests

Test Results

Inputs

Functional Configuration Audit

Verify that the CIs have satisfied their specified requirements

Supplier Acquirer

Physical Configuration Audit
CM Audits (continued)

Physical Configuration Audit (PCA)

◆ A formal examination of the as-built physical configuration of CI products against their design documentation \(^1, 2\)

◆ This establishes the Product Baseline

◆ This audit is conducted by the Supplier for the Acquirer and attended by
  – Management
  – System Engineering
  – Hardware / Software Engineering
  – Test Engineering
  – QA and CM
  – Contracts

of both the Acquirer and Supplier
CM Audits (completed)

PCA (completed)

Supplier As-Built Products:
• Design Documentation
• Code
• Hardware
• Etc.

Design Documentation

Physical Configuration Audit
Examination of the “as-built” configuration of CIs against their documentation
Supplier
Acquirer

Outputs
Product Baselines

Implemention

Physical
Configuration Management in Detail

- Configuration Management Planning
- Configuration Identification
- Configuration Control
- Configuration Management Audits
- Configuration Status Accounting
CM Concepts in Detail (continued)

Configuration Status Accounting

◆ The Configuration Status Accounting (CSA) task gathers, correlates, maintains, and provides status on CM controlled products and CM tasks.

◆ Provides the means for reporting status on:
  - Configurations
    ◆ FCI
    ◆ ACI
    ◆ PCI
  - Baselines
    ◆ FBL
    ◆ ABL
    ◆ PBL
  - Other
    ◆ CM metrics
    ◆ CM activities
    ◆ CM Audits

◆ Conducted by both the Supplier and the Acquirer
Configuration Status Accounting (concluded)

Supplier
- Configuration Status Accounting Reports
  produced by the CM organization
- Management and Staff

Acquirer
- Monthly Reports
- Program Management Reviews
- Milestone Reviews

AI Florence MITRE
Where are we?

- Introduction
- Configuration Management Concepts
- Configuration Management in Detail
- Tailoring Configuration Management
- Points to Remember
- References / Suggested Reading
- Questions / Answers / Discussion
- Contact Information
The shoe “gotta” fit to be comfortable

Tailoring Configuration Management

Big Project

Small Project

Big CM Wrong CM

Small CM Wrong CM

CM

Project

YES!

NO!

Al Florence MITRE
Tailoring CM (continued)

◆ CM can be very dangerous if under or over applied
  – Too much CM can stifle projects with bureaucracy
  – Too little CM will result in the projects getting out of control

◆ No CM will result in late deliveries, cost overruns, poor reliability, and even total failure

◆ CM needs to be tailored and appropriately applied to the scope of the application; the following are some factors to consider when tailoring:

  – Cost
  – Schedule
  – Function
  – Performance
  – Safety
  – Security
  – Criticality
  – Reliability
  – Size of Application
  – Number of Suppliers
  – Relationship of Supplier / Acquirer
  – Number of Staff
Tailoring CM (concluded)

- For large, complex, critical projects, CM needs to be applied to its fullest formal extent.
- For smaller, less complex projects, CM may need to be tailored by analyzing the tailoring factors.
  - For example, a TRB may not be necessary.
- For small, non-complex projects (6 or fewer staff members, $500,000, 6 months) a formal CCB may not even be necessary.
- The important point is to apply the concepts and principles of CM as appropriate and necessary.
Where are we?

- Introduction
- Configuration Management Concepts
- Configuration Management in Detail
- Tailoring Configuration Management
- Points to Remember
- References / Suggested Reading
- Questions / Answers / Discussion
- Contact Information
Points to Remember

- CM is important to ensure that current configurations of items are known throughout their lifecycle and that changes to those configurations are managed and controlled.
- CM starts early in the development lifecycle and continues until the system is removed from operation.
- Configuration items are baselined at a specific time in their lifecycle as a reference point for change control.
Points to Remember

- Impact assessments must be conducted on CRs against function, cost, schedule, interface by all affected entities.
- CM needs to be tailored and appropriately applied to the scope of the application.
- CM relationships and responsibilities between the Acquirer and Supplier must be understood and adhered to.
- All organizations have CM roles and responsibilities which need to be appropriately applied if CM is to be successful.
References / Suggested Reading


References / Suggested Reading


Questions / Answers
Discussion
Contact Information

Al Florence
Florence@MITRE.org
(703) 983-7476
GRAND SYSTEMS
DEVELOPMENT TRAINING
PROGRAM
VERSION 10.1

The union of system engineering, domain engineering, functional management, and program management for the greater good of the enterprise and customer base.

VOLUME 2R
AN EFFECTIVE SPECIFICATION
DEVELOPMENT ALGORITHM

Presented By
Jeffrey O. Grady

JOG SYSTEM ENGINEERING, Inc.
6015 Charae Street
San Diego, California 92122
(858) 458-0121
(858) 456-0867 Fax
jgrady@ucsd.edu or jeff@jogse.com
http://www.jogse.com

Copyright 2006

No part of this manual may be scanned or reproduced in any form without permission in writing from the author.
<table>
<thead>
<tr>
<th>PARAGRAPH</th>
<th>TITLE</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Specification Templates and DIDs</td>
<td>2</td>
</tr>
<tr>
<td>1.1</td>
<td>Enterprise Engineering Work</td>
<td>2</td>
</tr>
<tr>
<td>1.2</td>
<td>System Engineering Generic Work</td>
<td>3</td>
</tr>
<tr>
<td>1.3</td>
<td>Proposal Work That Prepares for Program Execution</td>
<td>4</td>
</tr>
<tr>
<td>1.4</td>
<td>Work Subsequent to Contract Award</td>
<td>6</td>
</tr>
<tr>
<td>1.5</td>
<td>The Preferred Templates</td>
<td>7</td>
</tr>
<tr>
<td>1.6</td>
<td>Modeling Work Product Capture Document</td>
<td>9</td>
</tr>
<tr>
<td>2</td>
<td>Structured Analysis</td>
<td>9</td>
</tr>
<tr>
<td>2.1</td>
<td>Traditional Structured Analysis</td>
<td>9</td>
</tr>
<tr>
<td>2.1.1</td>
<td>A System Defined</td>
<td>11</td>
</tr>
<tr>
<td>2.1.2</td>
<td>The System Environment</td>
<td>11</td>
</tr>
<tr>
<td>2.1.3</td>
<td>System Functionality</td>
<td>12</td>
</tr>
<tr>
<td>2.1.4</td>
<td>Performance Requirements Derivation and Allocation</td>
<td>15</td>
</tr>
<tr>
<td>2.1.5</td>
<td>Product Entity Structure</td>
<td>15</td>
</tr>
<tr>
<td>2.1.6</td>
<td>Allocation Pacing Alternatives</td>
<td>17</td>
</tr>
<tr>
<td>2.1.7</td>
<td>System Relations</td>
<td>18</td>
</tr>
<tr>
<td>2.1.8</td>
<td>Environmental Relation Algorithm</td>
<td>20</td>
</tr>
<tr>
<td>2.1.8.1</td>
<td>System Environmental Relations</td>
<td>20</td>
</tr>
<tr>
<td>2.1.8.2</td>
<td>End Item Service Use Profile</td>
<td>20</td>
</tr>
<tr>
<td>2.1.8.3</td>
<td>Component Environmental Relations</td>
<td>21</td>
</tr>
<tr>
<td>2.1.8.4</td>
<td>Environmental Impact</td>
<td>22</td>
</tr>
<tr>
<td>2.1.9</td>
<td>Specialty Engineering and RAS Complete</td>
<td>22</td>
</tr>
<tr>
<td>2.1.10</td>
<td>RAS-Complete in Table Form</td>
<td>24</td>
</tr>
<tr>
<td>2.1.11</td>
<td>Traditional Structured Analysis Summary</td>
<td>25</td>
</tr>
<tr>
<td>2.1.12</td>
<td>SDD Content and Format</td>
<td>26</td>
</tr>
<tr>
<td>2.1.12.1</td>
<td>Document Main Body</td>
<td>27</td>
</tr>
<tr>
<td>2.1.12.2</td>
<td>Appendix A, Functional Analysis</td>
<td>27</td>
</tr>
<tr>
<td>2.1.12.3</td>
<td>Appendix B, System Environment Analysis</td>
<td>27</td>
</tr>
<tr>
<td>2.1.12.4</td>
<td>Appendix C, System Architecture Analysis</td>
<td>28</td>
</tr>
<tr>
<td>2.1.12.5</td>
<td>Appendix D, System Interface Analysis</td>
<td>28</td>
</tr>
<tr>
<td>2.1.12.6</td>
<td>Appendix E, Specialty Engineering Definition Analysis</td>
<td>28</td>
</tr>
<tr>
<td>2.1.12.7</td>
<td>Appendix F, System Process Analysis</td>
<td>28</td>
</tr>
<tr>
<td>2.1.12.8</td>
<td>Appendix G, Requirements Analysis Sheet</td>
<td>28</td>
</tr>
<tr>
<td>2.1.13</td>
<td>Team Activity During Requirements Work</td>
<td>29</td>
</tr>
<tr>
<td>2.2</td>
<td>UML</td>
<td>30</td>
</tr>
<tr>
<td>2.2.1</td>
<td>Entry Analysis and Overview</td>
<td>30</td>
</tr>
<tr>
<td>2.2.2</td>
<td>The Connection Between Modeling Artifacts, Specification Content,</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td>and Product Entities</td>
<td></td>
</tr>
<tr>
<td>2.2.3</td>
<td>Dynamic Modeling Artifacts Explained</td>
<td>35</td>
</tr>
<tr>
<td>2.2.3.1</td>
<td>Sequence Diagram</td>
<td>35</td>
</tr>
<tr>
<td>2.2.3.2</td>
<td>Communication Diagram</td>
<td>37</td>
</tr>
<tr>
<td>2.2.3.3</td>
<td>Activity Diagram</td>
<td>37</td>
</tr>
</tbody>
</table>
TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>PARAGRAPH</th>
<th>TITLE</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.2.3.4</td>
<td>State Diagram</td>
<td>38</td>
</tr>
<tr>
<td>2.2.4</td>
<td>Structural Analysis</td>
<td>39</td>
</tr>
<tr>
<td>2.2.4.1</td>
<td>The Class</td>
<td>40</td>
</tr>
<tr>
<td>2.2.4.2</td>
<td>Class Relationships</td>
<td>41</td>
</tr>
<tr>
<td>2.2.4.3</td>
<td>Messages</td>
<td>42</td>
</tr>
<tr>
<td>2.2.5</td>
<td>Related Analyses</td>
<td>42</td>
</tr>
<tr>
<td>2.2.5.1</td>
<td>Specialty Engineering</td>
<td>42</td>
</tr>
<tr>
<td>2.2.5.2</td>
<td>Environmental Requirements</td>
<td>42</td>
</tr>
<tr>
<td>2.2.6</td>
<td>Specification Structure</td>
<td>42</td>
</tr>
<tr>
<td>2.2.7</td>
<td>Software Requirements Close-out</td>
<td>44</td>
</tr>
<tr>
<td>2.3</td>
<td>Opening the Analysis With DoDAF</td>
<td>45</td>
</tr>
<tr>
<td>2.4</td>
<td>Integrated Modeling</td>
<td>47</td>
</tr>
<tr>
<td>3.1</td>
<td>Summary of Team Activity During Requirements Work</td>
<td>51</td>
</tr>
<tr>
<td>3.2</td>
<td>Requirements Tools Base</td>
<td>52</td>
</tr>
<tr>
<td>3.3</td>
<td>Recommended Specification Responsibility Pattern</td>
<td>53</td>
</tr>
<tr>
<td>3.4</td>
<td>Requirements Risk Management</td>
<td>54</td>
</tr>
<tr>
<td>3.4.1</td>
<td>Requirements Validation</td>
<td>54</td>
</tr>
<tr>
<td>3.4.2</td>
<td>Margins and Budgets</td>
<td>55</td>
</tr>
<tr>
<td>3.4.3</td>
<td>Risk Tracking</td>
<td>55</td>
</tr>
<tr>
<td>3.5</td>
<td>Verification Requirements</td>
<td>58</td>
</tr>
<tr>
<td>3.6</td>
<td>Specification Review and Approval</td>
<td>59</td>
</tr>
</tbody>
</table>

A  APPENDIX A, PRESENTATION MATERIALS               A-i
B  APPENDIX B, SPECIFICATION DATA ITEM DESCRIPTIONS | B-i
    JOGSE System Specification Data Item Description | B-1-1
    JOGSE Hardware Item Performance Specification Data Item Description | B-2-1
    JOGSE Software Requirements Specification Data Item Description | B-3-1

NOTE

Exhibit B available from the lecturer by sending an email to jgady@ucsd.edu and requesting it.
<table>
<thead>
<tr>
<th>FIGURE</th>
<th>TITLE</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Overall Process</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>Preparatory Steps</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>Proposal Team Work</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>Program Work</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>System and Hardware Specification Template</td>
<td>8</td>
</tr>
<tr>
<td>6</td>
<td>Overview of the Traditional Structured Analysis Process</td>
<td>10</td>
</tr>
<tr>
<td>7</td>
<td>Ultimate System Diagram</td>
<td>11</td>
</tr>
<tr>
<td>8</td>
<td>System Environment</td>
<td>12</td>
</tr>
<tr>
<td>9</td>
<td>System Context Diagram</td>
<td>12</td>
</tr>
<tr>
<td>10</td>
<td>Function Sequence</td>
<td>13</td>
</tr>
<tr>
<td>11</td>
<td>Function Decomposition</td>
<td>14</td>
</tr>
<tr>
<td>12</td>
<td>System Life Cycle</td>
<td>14</td>
</tr>
<tr>
<td>13</td>
<td>Traditional Requirements Analysis Sheet</td>
<td>15</td>
</tr>
<tr>
<td>14</td>
<td>Product Entity Structure</td>
<td>16</td>
</tr>
<tr>
<td>15</td>
<td>Juxtaposition of RAS and N-square Diagrams</td>
<td>18</td>
</tr>
<tr>
<td>16</td>
<td>A Geometric View of the RAS Complete</td>
<td>23</td>
</tr>
<tr>
<td>17</td>
<td>RAS-Complete in Tabular Form</td>
<td>25</td>
</tr>
<tr>
<td>18</td>
<td>The System Relationship</td>
<td>26</td>
</tr>
<tr>
<td>19</td>
<td>SDD Structure</td>
<td>27</td>
</tr>
<tr>
<td>20</td>
<td>Context Diagram</td>
<td>30</td>
</tr>
<tr>
<td>21</td>
<td>Unified Modeling Language Overview</td>
<td>31</td>
</tr>
<tr>
<td>22</td>
<td>Hierarchical Relationship Between UML Dynamic Modeling Artifacts</td>
<td>33</td>
</tr>
<tr>
<td>23</td>
<td>Sequence Diagram Example</td>
<td>36</td>
</tr>
<tr>
<td>24</td>
<td>Communication Diagram Example</td>
<td>37</td>
</tr>
<tr>
<td>25</td>
<td>Activity Diagram Example</td>
<td>38</td>
</tr>
<tr>
<td>26</td>
<td>State Diagram Example</td>
<td>39</td>
</tr>
<tr>
<td>27</td>
<td>The UML Classifiers</td>
<td>40</td>
</tr>
<tr>
<td>28</td>
<td>Structural Relationships</td>
<td>41</td>
</tr>
<tr>
<td>29</td>
<td>Association Adornments</td>
<td>41</td>
</tr>
<tr>
<td>30</td>
<td>Software requirements specification template</td>
<td>43</td>
</tr>
<tr>
<td>31</td>
<td>Evolving Product Entity Structure</td>
<td>45</td>
</tr>
<tr>
<td>32</td>
<td>DoDAF Development Sequence</td>
<td>46</td>
</tr>
<tr>
<td>33</td>
<td>Requirements Traceability Across the Gap</td>
<td>49</td>
</tr>
<tr>
<td>34</td>
<td>Modeling Over the Years</td>
<td>50</td>
</tr>
<tr>
<td>35</td>
<td>The Approaching Merge</td>
<td>51</td>
</tr>
<tr>
<td>36</td>
<td>Tools Environment</td>
<td>53</td>
</tr>
<tr>
<td>37</td>
<td>Risk Matrix</td>
<td>56</td>
</tr>
<tr>
<td>38</td>
<td>Program Risk Tracking Chart</td>
<td>56</td>
</tr>
<tr>
<td>39</td>
<td>Verification Traceability</td>
<td>57</td>
</tr>
<tr>
<td>40</td>
<td>Specification Review and Approval</td>
<td>59</td>
</tr>
<tr>
<td>TABLE</td>
<td>TITLE</td>
<td>PAGE</td>
</tr>
<tr>
<td>-------</td>
<td>--------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>1</td>
<td>Independent and Combined SDD Appendices</td>
<td>44</td>
</tr>
<tr>
<td>2</td>
<td>Risk Probability of Occurrence Criteria</td>
<td>56</td>
</tr>
<tr>
<td>3</td>
<td>Risk Effects Criteria</td>
<td>56</td>
</tr>
<tr>
<td>4</td>
<td>Program Risk List</td>
<td>57</td>
</tr>
</tbody>
</table>
An Effective Specification Development Algorithm

The first order of business on any new program is the identification of requirements that collectively define the problem to be solved. This work can be completed in a timely and affordable way producing a quality definition of the problem space in terms of the minimum collection of requirements each one of which defines an essential characteristic of the system, or item thereof, to be developed. The target we should shoot for is all of the essential characteristics identified (no missed essential characteristics) and no unessential characteristics identified. Success in this process can be encouraged by the enterprise developing a set of specification templates and a corresponding set of data item descriptions coordinated with models selected to accomplish the requirements work and an effective suite of tools within which to capture the results. Every requirement that appears in every specification should be traceable to a model artifact from which it was derived. Before embarking on a new program, the enterprise needs to have selected this preferred set of models and tools and trained its staff to employ those models effectively entering the results in the tools suite. Every specification released is printed from the tools suite and should pass through a formal review and approval process as should any subsequent changes to any specification. As the models create work products, commonly simple diagrams, they should be saved in an organized fashion in paper or computer medias and formally released so as to be available for future system phases and modification projects.

Figure 1 offers a view of the overall process within which the preparatory work and definition activity that is discussed in the first section falls. Before entering a program, the enterprise should have prepared itself for the requirements work that will have to be done. The enterprise needs models to apply for the cases where the product is going to be implemented in computer software and in hardware. Both cases are covered in this tutorial followed by a discussion of integrated modeling.
1. Specification Templates and DIDs

Many system development organizations experience some difficulty in clearly identifying appropriate requirements for inclusion in specifications they must develop and they find it difficult to accomplish the work in an affordable and timely fashion. Over a period of the last two years the author developed an algorithm for improving system development organization ability in this work using templates and specially developed data item descriptions (DIDs). It requires some work to prepare the functional organization to support programs and some work on the part of proposal teams to accomplish initial analyses and extend the templates made available from the functional system engineering department to provide program-ready data, and work by the program teams starting with contract award and running through the period of time while specifications are being developed.

The goal of this specification algorithm is to provide for affordable and timely enterprise and program definition and documentation of new product technical requirements, the management and maintenance of related data, and publication and subsequent configuration control of the resulting documents.

The work required to implement the algorithm can be described in three preparatory steps: (1) enterprise engineering work, (2) system engineering generic work, and (3) proposal team work. The first two steps are illustrated in Figure 2. The numbers in the blocks coordinate with the steps in the algorithm. Recommended functional department responsibility for accomplishing the indicated tasks is noted at the lower left corner of the task blocks.

![Figure 2 Preparatory Steps](image)

1.1 Enterprise Engineering Work

Identify and staff an enterprise integration team (EIT) that is responsible for engineering the enterprise common process and acting as the process integration and optimization agent during its development and implementation on programs. The EIT should report to the enterprise executive.
1.1.1 The enterprise, through the efforts of the EIT, must develop a common process diagram that generically identifies all program work at a level of indenture that is adequate for making clear what work must be commonly done and allocating the corresponding work responsibilities to functional departments responsible for supplying programs with the necessary resources to accomplish that work well.

1.1.2 Allocate all work on the common process diagram to functional departments forming a task allocation matrix. This matrix establishes the requirements work that functional departments must be prepared to do on programs and any training that the functional departments are funded for and capable of performing must be focused on these tasks. This matrix covers the whole enterprise capability but in this section we are focused on doing the requirements work.

1.1.3 Functional departments collect all work allocated to them and build department manuals that explain what work must be done and provide links or descriptions for how to perform this work. One of the departments will be system engineering that will have responsibility for specification development and management on programs. For each task a functional department has responsibility, that department must identify work products that will result as a function of having accomplished the work on a program. EIT must integrate and optimize the evolving functional department work descriptions and work product identifications to ensure overall efficiency and effectiveness. All work products must have at least one user. Work products must be linked to the common process diagram tasks. Specifications are an example of a task work product and the work product of interest in this algorithm.

1.2 System Engineering Generic Work

1.2.1 To the extent that work products are documents, the responsible functional department must prepare a template containing the basic structure of the document in terms of generic paragraphing structure and calls for graphical images. In preparing for implementation of specification development and management work on programs in general, the functional system engineering department will select the specification standards that will be applied respecting the common customer base of the enterprise. They will review these standards and associated data item descriptions (DID), ensure that the system engineering department manual adequately covers specification standards the enterprise has chosen to respect, and build a set of specification templates (paragraph numbering and titles only), one for each kind of specification that will commonly have to be prepared on programs.

1.2.2 For each specification template defined, the system engineering department will determine one or more preferred modeling approaches for each kind of requirement in the template. The modeling approaches encouraged are the following relative to the primary kinds of specifications that will have to be developed:

- System Specification: Traditional Structured Analysis
- Hardware Performance Specification: Traditional Structured Analysis
- Software Performance Specification, Database: IDEF-1X
- Software Performance Specification, DoD IS: DoDAF
1.2.3 Map models to templates such that for any of the specifications listed above there is a model identified for each paragraph of each specification. Ideally, all of the paragraphs of a particular kind of specification would employ models from the same family as suggested in the pairing above.

1.2.4 Map specialty engineering disciplines to specification templates in preparation for program mapping of these disciplines to specific product entities as a means of directing specialty engineering requirements analysis work.

1.2.5 For each template and model combination, prepare a DID communicating how the analyst will prepare the specification using a particular modeling approach and template. The DID must clearly show the connection between the modeling artifacts and the template paragraphing structure. The DID paragraphing structure must coordinate with the modeling components that are intended to yield derived requirements.

1.2.6 Map the functional departments that will be responsible for performing requirements analysis on programs to the template paragraphing structure for each kind of specification telling where the programs will acquire the analysts to accomplish the specification development work.

1.2.7 Prepare a template and DID for a system definition document (SDD) within which program structured analysis work products will be captured and configuration managed. These work products are to be captured in appendices of the document. An alternative is to capture the modeling artifacts within a computer tool that can be configuration managed.

1.2.8 Map the appendices of the SDD to the paragraphing structure of the templates telling in what appendix the corresponding work products will be captured.

1.2.9 Combine the functional department map (1.2.5), models map (1.2.3), and SDD appendix map (1.2.7) on a single matrix for each template and make these matrices available for program use.

1.2.10 EIT and the functional system engineering department must cooperate on selection of one or more computer tools or paper and pencil algorithms with which to accomplish requirements analysis on programs. Built a generic schema coordinated with preferred methods and models.

1.3 Proposal Work That Prepares the Program for Initiation

Proposal team work is illustrated in Figure 3. Blocks that do not have numbers coordinating them with the steps in this algorithm are not covered by the algorithm because they are not directly related to requirements analysis and specification development but these blocks add valuable context.
1.3.1 When beginning the proposal or program work, the manager should establish a program integration team (PIT) staffed by engineering, manufacturing, verification, logistics, and quality and a program business team (PBT) staffed by finance, contracts, scheduling, business information systems, and administration. Both of these teams should report to the program manager. These two teams could be combined as a staff function to the proposal manager but they will have integrating and optimization roles to play across the product oriented teams.

1.3.2 The PIT will perform an initial system analysis that will result in a clear understanding of any requirements provided by the customer, formatting of those requirements into alignment with the enterprise DID for a system specification, and adding to those requirements the results of their own system analysis work. A system environmental requirements analysis activity will expose a set of tailored standards covering the natural environment corresponding with spaces within which the system shall have to operate. A threat analysis will lead to exposure of hostile requirements. An interface analysis will identify external and top level internal interfaces that will be characterized in requirements statements. The result will be a preliminary system specification for submission with a proposal. If possible, this analysis work will be continued to develop all of the immediately subordinate specifications each of which will be the development responsibility of one of the top level integrated product and process teams (IPPT) to be identified and staffed subsequent to contract award.

1.3.3 The modeling work described in paragraph 1.3.2 will yield modeling artifacts from which requirements may be derived. The preliminary system specification development and any other specifications developed during the proposal effort will follow the pattern described in the
The second tier specification development may be delayed until a contract award but should precede the formation of the top tier IPPT. In all cases, requirements are derived from a model.

1.3.4 Requirements flowing from the structured analysis work will flow into a requirements analysis sheet (RAS) implemented in a computer database tool. All requirements entered into the RAS must include a traceability reference to the model from which they were derived.

1.3.5 Out of the initial PIT system analysis work will also come the preferred product entity breakdown diagram upon which the PIT, working in concert with integrated business team personnel, will construct overlays for organization responsibility breakdown (IPPT assignments), specification tree breakdown, engineering drawing breakdown, work breakdown (WBS), and manufacturing breakdown. The work breakdown will be handed off to the business team that will use it as the basis for building the program work definition, cost estimate, and IPPT work budgets. The IPPT will be assigned so as to align perfectly with the WBS making it possible to present to each IPPT leader as the teams are formed, a copy of the top level specification for which the team is responsible and the related budget and planning package for the whole WBS the team is responsible for as well as their top level schedule responsibilities encouraging the result that the IPPT leader may be held accountable for managing all aspects of the development of the entity assigned to the team.

1.3.6 The PIT will select a set of templates that correspond with the kinds of specifications that will have to be prepared on the program and the related DIDs that are coordinated with the models that will be applied in the analytical work. The PIT must also map specialty engineering disciplines to product entities to aid teams being formed to staff appropriately.

1.3.7 The PIT must take action to cause adequate computer tool seats to be allocated to the proposal team and accomplish any planning necessary for the subsequent program relative to the use of any requirements database tools and make any needed adjustments in database schema for the program.

1.3.8 The PIT shall capture the results of structured analysis work performed during the proposal activity in a preliminary system definition document (SDD) that will be used as the basis for subsequent lower tier analyses.

1.3.9 Any specifications developed in the proposal effort must be formally reviewed and approved by the proposal manager.

1.4 Work Subsequent to Contract Award

Specification related work to be accomplished subsequent to contract award is illustrated in Figure 4. This work is repetitious in nature progressively working down through the expanding architecture. Top-level teams may shred out during program work yielding sub-teams but in all cases, the top-level teams are responsible as managers for all lower tier team activity. This telescoping management responsibility is applicable throughout the team structures. During program performance, lower tier requirements analysis responsibility may be passed down
through the team structure with immediately superior team reviewing and approving of all immediately lower tier team specifications or the responsibility may be retained by the higher-level team but these decisions must be coordinated with the team budgets and staffing considerations arrived at during proposal work.

Figure 4  Program Work

1.5 The Preferred Templates

Ideally, the development organization would build a set of templates (paragraphing structure with no content) and data item descriptions (DID) that tell how to build a specification following the related template using a particular set of models. These should be maintained by the functional department in system engineering that has overall requirements and specification work responsibility. These should be available for reference or download by any new program.

Figure 5 offers a view of the preferred template for a system or hardware specification using traditional structured analysis as the modeling choice. The template is annotated with the preferred modeling artifact that will be used to identify the corresponding requirements and the functional department from which the program will obtain personnel to accomplish the related modeling work using the Figure 3 organizational structure. The data item description (DID) acronym in the model column means that the content is driven by the content of the DID used as the basis for the specification. The department references are cut at a very high level in this case and should be identified at one or two layers below this level but Figure 3 goes no deeper. The APP column gives the Appendix in the System Definition Document where the work products can be found.

The structure in Figure 5 can also be used for computer software requirements specifications (SRS) with paragraph 3.1 rewritten for UML and the model column updated to reflect UML artifacts.
A similar mapping should be provided for the software requirements specification (SRS) based on, in the author’s view, the use of UML. Customers often require conformance to a DID supplied by them but may permit tailoring. The outline included in Figure 5 is a significantly tailored version of MIL-STD-961E to group all requirements so as to correspond with the modeling components contained in traditional structured analysis described in paragraph 2.2.1. k of the standard. The author believes this format will work with software as well but a different DID is required coordinated with the modeling approach selected (UML encouraged).
1.6 Modeling Work Product Capture Document

Programs should also be provided with a template for a System Definition Document (by whatever name) within which they can easily capture the results of all modeling work so that it can be preserved beyond the period of time when that work is actively being pursued. A later section describes this document. The appendices of the SDD are referred to in Figure 5 in the APP column and explained in Figure 20 and related text.

2 Structured Analysis

There are many models that can be used to accomplish an organized requirements identification effort that is preferred to an ad hoc method because it will most often hit the target noted earlier - identification of all of the essential characteristics and identification of no unessential characteristics. This process description encourages the use of traditional structured analysis in the near term to develop and identify the requirements for systems, hardware, facilities, real property improvements, and personnel actions captured in procedures as discussed in paragraph 2.1. One of the most difficult tasks in system development revolves around the relations between the system entities, the interfaces. This discussion of traditional structured analysis also contains a complete algorithm for identifying all system relations. Where the product is going to be implemented in computer software, unified modeling language (UML) is encouraged within a process context offered in paragraph 2.2. It is intended that a development organization should develop a process transformation roadmap needed to transition from this mixed method for the modeling work to a universal modeling approach, still evolving, that can be applied to all requirements work at the earliest possible. See paragraph 2.3 for a discussion of integrated modeling.

2.1 Traditional Structured Analysis

Figure 6 illustrates an overview of the traditional structured analysis process (TSA). The eleven numbered steps are briefly introduced followed by additional details in subordinate paragraphs.

1. Understand the User Requirements - Through conversation with the user and/or reading a user requirements statement or specification, the developer tries to understand the user's need. This is not ever easy because the user is able to explain only what their mission interests are and the developer needs hard engineering data.

2. Decompose - Users commonly have problems that are too grand to be easily understood in a single small document or simple diagram. These problems commonly have to be decomposed or partitioned into a collection of smaller related problems.

3. Functional Flow Diagram - The TSA approach employs some form of functional analysis as the decomposition medium.

4. Performance Requirements Analysis - The functions are translated into performance requirements.

5. Requirements Analysis Sheet (RAS) - The strings of functions, performance requirements, and product entities to which they are allocated appear as rows in the RAS. The RAS also is used to capture all system requirements linked to the model from which they were derived.
6. Requirements Allocation - Performance requirements are allocated to product entities in the RAS.
7. Product Entity Structure - The physical and logical entities that comprise the system are arranged in a hierarchical structure that is used as the basis for WBS, specification identification, team assignments, and many other program applications.
8. N-Square Diagram - The interfaces that must exist between the product entities are identified through a pair-wise analysis of all possible interface relationships.
9. Environmental requirements for the expanding product entities are determined through application of a three-layer model.
10. Specialty engineering requirements are identified by a group of specialist linked to the product entity structure by a specialty engineering matrix.
11. This process is applied iteratively as lower tier entities are identified through functional analysis.

Figure 6  Overview of the Traditional Structured Analysis Process
2.1.1 A System Defined

A man-made system is a collection of entities that are meant to interact in predictable ways with an environment and with each other via relations between them to achieve a useful function identified and articulated by a customer as a mission need statement. Therefore, systems are composed of entities and relations between the entities. The system is intended to satisfy the mission need statement, the system’s ultimate function, depicted on system diagrams as a rectangular block titled System Need and identified with a functional identifier F. The need is allocated to the system depicted on system entity model by a rectangular block named “system” (or a particular system) and identified with a product entity identifier A.

A system interacts within an environment as shown in Figure 7. The environment for every system is everything in the Universe (U) less the entities that are part of the system product entity structure (Q = U - A). One can reduce the scope of the environment to those elements that will have some influence on the system. The line joining the system and environment in Figure 7 (I2) indicates the relations between the two (external interfaces). The line joining the system on both terminals (I1) indicates the internal relations between system entities (internal interfaces) yet to be defined within the system.

![Figure 7 Ultimate System Diagram](image)

2.1.2 The System Environment

The environment for any system is composed of the subsets illustrated in Figure 8. While all environmental effects on the system are relations, they may be partitioned between those that are commonly considered environmental stresses and the cooperative environmental elements that are treated as external interfaces commonly developed by a pair of teams or contractors responsible for the terminal product entities.

A context diagram, such as that shown in Figure 9, even though similar in nature to Figure 7, offers a useful simple model for focusing attention on identifying all external relations. Some of these terminators will be natural, non-cooperative, or induced environmental stresses. Others include hostile stresses determined through a system threat analysis as well as both stresses and useful relations with cooperative systems. Though this diagram was conceived as the beginning of the modern structured analysis model, it has a useful purpose in TSA as a means of viewing all of the inside-outside relationships and in UML as an organizer of use cases. It can be used to make a first impression in the line I2 in Figure 7.
The system natural environment is determined by defining all of the spaces within which the system will be employed based on an analysis of the intended mission and basing concept. The spaces are coordinated with a set of environmental standards. Each standard is studied for necessary content and the remainder tailored out. Each selected parameter is then studied for an appropriate range. The system natural environment is then the union of the selected parameters from the selected standards.

The non-cooperative environment is defined by determining what stresses will be applied to the system from man made systems which are neither hostile nor cooperative. An example of non-cooperative stress is electromagnetic energy. Self-induced environmental stresses are not easily
determined at the system level because one needs to understand energy sources and other stressors within the system determined as part of the design of end items.

System cooperative environmental relations are defined by determining how the system to be developed will associate with other friendly systems already in existence or being developed. These associations may be coupled into or out of the system in terms of information, physical association, materials, or energy.

2.1.3 System Functionality

A function is a necessary activity for a system to perform. It may be static, dynamic, or both. It should be named using an action verb followed by a noun. A function is depicted in modeling the system as a rectangular block identified by an action verb name centered in the box and a function identifier (ID) in the lower right corner. The ultimate function for any system is the customer need the name of which is the need statement possibly paraphrased to fit into the space provided coupled with a function identifier F.

Two or more functions can be linked together using directed line segments to show a sequence of functions. In Figure 10 the understanding is that function F1 must be accomplished before function F2. Combinatorial symbols may be added to permit more complex sequential relationships. The combinatorial symbols encouraged are AND, inclusive OR (IOR), and exclusive OR (XOR) with the common logical meanings. Enhanced functional flow block diagramming adds loop (repeat a function until a specific outcome has been attained) and iterate (repeat a function a specific number of times) combinatorial symbols that can be useful. Diagrams so constructed are called functional flow diagrams. These diagrams may be oriented on the page with their primary flow axis arranged horizontally or vertically with the flow in either direction.

![Figure 10 Function Sequence](image)

For any function with identifier F@ (where @ is a string of length n (including n=0) composed of characters from the set \{A through Z less O\}U(a through z less I)U(0 through 9)) there may exist one or more subordinate functions F@# (where # is a single character from the same set identified above which differentiates other functions at that level from one another). This is illustrated in Figure 11. Every function need not have an expansion. There is no need to assign function identifiers in alpha numeric sequence on a page of the diagram but it helps the human to use the diagram if they are assigned initially coordinated as much as possible with the function sequence. If a function is deleted subsequent to a release of the diagram, that identifier should not be used again. If the number of functions on any one diagram exceeds the maximum number of symbols available, 60, change the diagram to reduce the number to less than 60.
Ideally, who ever accomplishes the initial analysis of the need, would do so using the functional analysis process described here where the first decomposition is the system life cycle as shown in Figure 12 and the second is an expansion of the life cycle function “Use System” (F47) to expose the top level operational intent and initial content of the user requirements documentation or preliminary system specification. If the customer or other initial analysis agent applies an unstructured or ad hoc approach, then the development organization may have to accomplish a functional analysis and try to map the requirements identified by the customer (user and acquisition agent) to the functionality exposed when they do accomplish this work.
Ideally, the development organization would extend the functional analysis into the remainder of the system life cycle functions as well as the Use System function determining appropriate resources for the process steps of the development program and the product system being developed such that the physical product delivered will be jointly optimized relative to its product and process. Commonly, process functions do, or should, influence product functions and the corresponding product entities needed as well as the opposite case.

2.1.4 Performance Requirements Derivation and Allocation

The functions identified in the functional flow diagram must be translated into performance requirements that tell what the system and its parts must do and how well it must do them as shown in Figure 13. These statements can be first developed as primitive statements for example phrased as “Velocity ≥ 600 knots” without complete sentence structures and subsequently transformed into complete sentences in the chosen language. Traditionally, a requirements analysis sheet (RAS) has been used to capture the function identification, the primitive performance requirements statements, and the allocation of these performance requirements to product entities. One could allocate the function names directly to architecture but often one finds a one-to-many allocation result this way whereas allocation of performance requirements tends to follow a one-to-one pattern. To fully characterize a function it may require identification of multiple performance requirements and these several performance requirements may be allocated to different product entities.

The RAS as traditionally used is incomplete, unfortunately, and this discussion will use a RAS complete. We will show how the three kinds of constraints can also be captured in the context of Figure 13 shortly. The intent is to be able to capture all requirements in a RAS linked to a modeling artifact implemented in a computer application.

<table>
<thead>
<tr>
<th>FUNCTION MID</th>
<th>PRODUCT ENTITY PID</th>
<th>REQUIREMENTS RID</th>
<th>STATEMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>F123 Provide Thrust</td>
<td>A12 Engine</td>
<td>WEB9FS</td>
<td>Thrust &gt; 10,000 pounds at sea level</td>
</tr>
</tbody>
</table>

Figure 13 Traditional Requirements Analysis Sheet

2.1.5 Product Entity Structure

System functionality is accomplished by physical entities that form part of the physical system. These entities in the system, in the aggregate, comprise the product entity structure that the author formerly referred to as the system architecture. The word architecture has taken on a significantly broader meaning in recent years convincing the author that what he formally referred to as architecture should now be referred to by the more isolating term product entity
structure. The function $F$ is accomplished by product entity structure element $A$, the system, by definition as shown in Figure 7. Lower tier entities should be exposed following Sullivan’s encouragement of form follows function. Trade studies may be appropriate to make hard decisions on the best implementation of a particular function in early program phases. The physical entities that accomplish functionality can be partitioned into five classes: (1) hardware, (2) computer software, (3) facilities and improvements to real property, (4) procedural definition, and (5) humans following procedures or acting autonomously. We could merge the last two into one. The relationship between functions, the physical entities in the product entity structure, and the corresponding performance requirements is depicted on a simple requirements analysis sheet as shown in Figure 13.

The aggregate product entity structure for a system is illustrated in a hierarchical model connecting the product entities arranged into a breakdown block diagram as illustrated in Figure 14. The entity identifiers follow the same pattern as the function identifiers explained earlier beginning with a letter $A$ because of the author's previous use of the word architecture for this view. The entities stream out of the RAS available for structuring into the product entity block diagram. Ideally, this work would be accomplished by a team of people representing hardware and software engineering, manufacturing, procurement, and verification with strong system/program leadership by the PIT. Initial functional analysis and allocation must concentrate on understanding user mission needs. This will generally require intense interaction with the user, ideally using system models to encourage mutual understanding. Alternative ways of implementing functionality with different product entities should be considered and where the decision is very difficult they should be subjected to a trade study.

- **Figure 14** Product Entity Structure
2.1.6 Allocation Pacing Alternatives

The conduct of the functional analysis and allocation work can be paced in one of four different ways:

(1) Instant – As soon as functions and/or their corresponding performance requirements are identified, they must be immediately allocated to the expanding product entity structure.

(2) Terminal - All of the functional analysis must be complete before any functions and/or their corresponding performance requirements can be allocated to product entities.

(3) Layered - The analyst completes one layer of the expansion of a functions and must allocate all of the exposed functionality and/or the corresponding performance requirements to product entities before further expanding that function. This works best if all product entities related to a layer are defined in terms of their requirements and design concept before pursuing the next functional layer and its allocation but there generally is not sufficient time available and one must accomplish a lot of this work in parallel leading to some risk and interaction. The layered approach has been popularized by Mr. Bernard Morais and the late Dr. Brian Mar under the title FRAT.

(4) Progressive – The analyst completes several layers of the functional analysis without allocating any of it to product entities. At a layer guided by experience the analyst begins allocating performance requirements derived from higher tier functionality to product entities. Design concepts are defined for the product entities at the higher levels and these act to both guide and constrain lower tier functionality progressively. Allocation is delayed throughout the analysis such that higher tier design concepts help to steer lower tier functional analysis tending to isolate iteration to one structure (functionality, architecture, or allocation) at a time. The two extreme cases (1 and 2) are flawed due to a need to iterate F, A, and allocations excessively in the case of 1 and a common need to significantly change lower tier functionality after the higher tier allocations begin in the case of 2. The progressive approaches either by layer or guided by experience will generally produce better results with less modeling hysteresis.

There exists a downward limit in decomposition or expansion of the functional portrayal. This limit for any one branch in the expansion is best determined on the product entity plane based on the analyst’s understanding of the problems related to the lowest tier product entities. Where all of the lowest tier entities are fully characterized supporting procurement or in-house design, the functional analysis work can be reduced to maintenance of the models and related data. There is one more layer of requirements related to parts, materials, and processes but that is driven by design decisions during synthesis and commonly has already been completed by those responsible for the sources of these entities. The logistics analysis process may require a switch to a process diagram and if the progressive allocation approach described has been followed the functional flow will have been migrated toward a process diagram at the lower tiers.
Figure 15 illustrates a geometric view of the process of deriving performance requirements from functions and the allocation of those performance requirements to product entities. For example, a performance requirement has been derived from F4713 and allocated to product entity A11. This plane represents the normal RAS used only for function allocations. As suggested by the other structures we will use the geometric structure to expand the RAS to cover all requirements.

Figure 15  Juxtaposition of RAS and N-square Diagrams

2.1.7  System Relations

As the product entity structure is formed, the analyst can begin to identify the needed internal relations between the system entities by using an n-square diagram where the product entities are identified down the diagonal at some level of indenture. For a given analysis where the number of entities being studied for interface relations in an n-square diagram is N the largest possible number of relations is N(N-1) counting each direction as one possibility between each pair of entities. Interfaces between these entities is pre-determined by the way that functionality is allocated to the entities. Therefore, one may explore the list of function-product entity pairs
associated with each product entity in the n-square diagram. This is referred to as a pair-wise analysis of the n-square diagram intersections.

Figure 15 includes an n-square diagram with only half of the square showing (the remainder hiding behind the other structures in the diagram). The diagonal (containing the product entities we are interested in accomplishing the pair-wise analysis on) has been aligned with the product entity axis of the function-product entity matrix (which corresponds to the simple RAS of Figure 13).

The process for marking the intersections of the function driven relations matrix (n-square plane) entails a pair-wise analysis of the function-product entity pairing relationships marked on this matrix. Interface Ix is encouraged by the conclusion that if F4711 maps to A13 and F4712 maps to A11 then there is a possible demand for an interface between A11 and A13 to implement that relationship. So we map functions to product entities but we map F-A pairs to interfaces and those interfaces are pre-determined by the way we allocate functions to the product entities.

If the system is a modern war ship or the system that will take man to Mars, a pair-wise analysis of the function driven relations matrix would be beyond human comprehension if attempted all at one time. We can, however, partition the analysis work to one interface expansion at a time and it is not so overwhelming. At any one level of product entity granularity, we can explore one layer of product entity structure expansion for internal interfaces within the parent item. If there are five subordinate entities then the number of possible interfaces to be examined in a pair-wise fashion would be 5x4 or 20. Similarly, external interfaces can be analyzed individually. Of course, it will still be necessary to accomplish considerable interface integration work because of the partitioned process. This process will go very fast if the analyst is very familiar with the problem space and the evolving solution concepts. The complete algorithm is extended in subsequent paragraphs.

The requirements analysis sheet (RAS) identifies every possible pairing of functions and architecture entities. We may sort this listing so that all of the functions (or performance requirements derived from those functions) allocated to each entity are grouped by entity. Then we can pile up the allocations onto the product entity squares on the diagonal of a physical n-square diagram as suggested in Figure 15.

The discussion so far has dealt only with internal interface identification all defined on the function driven relations matrix of Figure 15. To cover external interfaces we add the larger n-square diagram on Figure 15. The diagonal in this case includes all of the product entities plus all of the external entities in the cooperative environment. We can identify relations between these external and internal entities in the same way we did the internal ones. The association of function F4712 to cooperative environment entity QC1 and allocation of a function to A12 may define a need for an interface I2y.
2.1.8 Environmental Relations Algorithm

2.1.8.1 System Environmental Relations

The system environment consists of all entities in the Universe less those that are in the system. That is, \( Q = (U - A) \) where \( Q \) is the environment, \( U \) is the Universe, and \( A \) is the product entity structure of the system being developed. It is convenient to partition all system level environmental relations into the sets illustrated in Figure 8. The system cooperative environment (QC) can actually be treated as an external interface and can be developed using the algorithm covered in Paragraph 2.1.7 very effectively. External cooperative systems are simply located on an extension of the product entity axis forming the cooperative environment axis. The hostile environment (QH) can best be understood through analysis of threats posed by hostile forces. The non-cooperative environment will yield to the same thought process applied in the threat analysis except that the stresses applied to the system are not applied for hostile purposes, rather simply because the system being developed is sharing a common operating space with other systems. Electromagnetic interference is an example of the stresses applied in this set.

System time (QN2) is studied using time lines oriented about the functions that the system must satisfy. When we allocate those function (or their corresponding performance requirements) to architectural entities the timing requirements corresponding to the functions are applied to the entity as timing requirements.

System space (QN1) is defined through mission analysis such that it is determined in what volumetric spaces of the Earth (surface, subsurface, and aerodynamic) and/or surrounding space and/or distant bodies the system shall function within, on, or around. For each space, that space is teamed up with one or more natural environmental standards that define that space. Each of these standards is then studied to determine which natural environmental (QN3) parameters included in the standard shall apply to the system being developed. Those that do not apply are tailored out of the standard. The selected parameters are then studied to ensure that the range of values is appropriate for the system. If the range in the standard is too broad it is tailored to narrow the range of values to that for which the system shall be designed. This process is repeated for each standard linked to a system operating space.

Figure 15 also extends the RAS concept to include environmental requirements analysis. The system environment as depicted in Figure 7 is illustrated at the diagram “origin” on the environmental axis that happens to coincide with the architecture “origin” that corresponds to the whole physical system A.

2.1.8.2 End Item Service Use Profile

An end item is a major element of a system that generally retains its physical configuration throughout its mission performance and has an end use function. The end items may remain fixed in place during use or move over great distances and maneuver within the system spaces as a function of the system mission and the end item’s application in the system. Each end item should be designed to endure only those natural environmental stresses anticipated so it is necessary to determine what subset of the system natural environment each end item shall be
exposed to. To accomplish this, one must create a physical process flow diagram. This is not the same as the functional flow diagram used to identify system architecture and performance requirements. The blocks on a functional flow diagram represent things that have to happen whereas the block of a process diagram represent a real world analogy. You cannot profitably consider system entities flowing through the functional flow diagram, indeed we are using the diagram to determine what those entities should be. However, we can imagine the system product entities flowing in the process diagram where each process acts as a transformation on the system entities.

The first step in defining end item environmental relations is to map the system environmental parameters to the process steps at some level of indenture, generally at a level where the environmental map does not change significantly below that process level. This work defines an environment for each process. The next step is to map the architectural entities to the process blocks. If an entity only maps to a single process step, it simply inherits the process environmental set. If, as is so often the case, the architectural entity maps to two or more process blocks, then it will be necessary to apply some kind of integrating process to any differences in environmental stresses and their values observed between the two or more process blocks. The rule most often selected initially is to pick the worst-case range for each parameter across the process values being evaluated. If this rule does not adversely influence system cost, then it is an adequate solution. If this approach either results in an adversely narrowed system solution space, then an alternative to “worst case” must be derived. Often time lines will show that the problem environmental stress will be applied over such a short time as to be insignificant. In other cases, one may find that the problem can be narrowed to some particular combinations of values that are very unlikely to occur. If the problem is intractable, it may be necessary to restrict one or more system environmental parameters more severely than is currently being done. Generally, this will have an adverse effect on system performance.

The self induced environment (QI) can best be studied and defined at the end item level since it is end items which contain the sources of energy and other stresses of interest which will reflect commonly through a natural environmental parameter right back into the system. Since the self-induced stresses are commonly greater in magnitude than the corresponding natural stresses for that same parameter, these induced relation values have the effect of extending the range defined through the application of the end item service use profile algorithm discussed above.

2.1.8.3 Component Environmental Relations

The environmental relations appropriate for components installed within end items are simply the end item environmental stresses if the end item has no altering effect on those stresses and all spaces within the end item offer the same environmental stresses to components installed within them. Where an end item does have a modifying effect on end item stresses but all spaces within and upon the end item offer the same stresses, it is necessary to determine the end item design effect on end item environmental stresses and the result is the set of installed component environmental stresses. The most complex case occurs when the end item must be partitioned into two or more spaces more often called zones of common environmental stresses. The value of each end item parameter must be determined for each zone thus defining the environment for each zone. Then one maps the components to the zones and they inherit the zone environments.
Commonly, the job is not complete at this point because it is found that there is no zone within which one or more components can be installed in a particular end item that will cause their environmental stress limits to be satisfied. When this happens, it is necessary to either change the component environmental specification values or include an environmental control system as an added entity into which the components with the environmental range shortfall problem are placed.

2.1.8.4 Environmental Impact

The environmental effects discussed in the three previous cases deal with environmental stresses the environment will apply to the system but there may be cases of the opposite direction that will cause damage to the environment. Once identified by someone skilled in environmental impact, these can be treated like safety hazards to be mitigated through re-design, procedural controls, or compensating environmental actions. In the case of military systems it is very difficult to mitigate the damaging effects of warfare but these systems can also have damaging effects on the environment in peaceful use such as training and maintenance. Often these materials are just naturally dangerous to be around as illustrated in the difficulties observed in the base closure efforts where many adverse environmental effects have had to be identified and mitigated.

2.1.9 Specialty Engineering and RAS Complete

The system engineering agent for the system must build a list of all of the specialty engineering disciplines that will be applied in the development of the system. A specialty engineering scoping matrix should be prepared between specialty engineering disciplines that may be required in development of the system and the product entities. This will help to determine team staffing needs in that area and connect people in those disciplines with a need to do specialty engineering requirements analysis for the indicated items. Figure 16 adds one more plane, the specialty engineering scoping matrix, to the construct previously illustrated in Figure 15 providing for allocation of specialty engineering disciplines listed to architecture.

Specialty discipline H7 is shown mapped to architecture item A11. This must be followed by analyst definition of one or more discipline H7 requirements that will flow into the specification corresponding to the product entity. The structure exposed in Figure 16 is a complete RAS showing all of the important requirements related relationships supporting the requirements analysis process leading to the identification of every kind of requirement appropriate to the system and hardware specifications and all of them linked to and derived from a model.
The fact that an aircraft airframe will have to be checked for alignment during manufacture and after hard use (hard landing or pulling excessive g’s in flight, for example) identifies a need for a relation between the airframe and the equipment which will be used to accomplish the alignment. Today this will generally call for some form of laser optical application so the airframe would have to either include targets, detectors, or mirrors or provisions for these to be applied. The manufacturing and maintenance engineers would have to consider all of the ways there may have to be relations between support equipment and the operational entity. There may be other cases where these disciplines have to call for internal relations within the system entities. For example, if the system must include built in test (BIT), there will have to be relations between most of the on board equipment and some entity that concentrates the BIT effects for display to operations and/or maintenance personnel.
The needs of operations personnel, such as aircraft pilots, locomotive engineers, ship captains, and automobile drivers provide a tremendously rich class of entity relation possibilities. The physical relations are always fairly simple in that the human operator has only his/her senses, mental acuity, and physical strength through which to interact with the system. This problem is made much more complex, however, because not all humans will react in precisely the same way to a particular stimulus. It will be necessary to determine the complete data set that the human will require under all operating conditions and in what way the human shall influence system behavior in terms of controls. Operator sequence diagrams, built like a UML activity diagram with swim lanes, can be useful in doing this work.

Every one of the specialty engineering disciplines selected for the program must be evaluated for entity relation driving potential and those persons doing that work alerted to their responsibilities in identifying relations for further consideration by the whole team.

2.1.10 RAS-Complete in Table Form

The results from the analyses noted in prior text must be captured on its way into program specifications. Certainly, the most advantageous way to do that is in a computer database systems such as DOORS, CORE, or SLATE. Therefore we would expect that some form of tabular structure would suffice. Figure 17 is offered as the candidate view of this table for use during development in capturing the relationship between model, requirements, product, and document entities. The model ID (MID) identifies the model from which the requirement was derived. The requirement columns identify the requirement ID (RID) assigned by the computer system for use in traceability. The RID is a made-up computer-assigned unique field using a base 60 numbering system in this example but a hexa-decimal implementation is probably more common. Ideally the requirement statement should be captured in primitive form (attribute, relation, value, and units) wherever possible with different fields for each component of the string. The primitive form is shown concatenated in Figure 17. The final column pair offers specification paragraph number and title.

The sample data included is ordered by model ID alphanumerically separating the data into the four kinds of requirements found in a system or hardware specification. The lists the MIDs respected by the author is still in a state of change as different RAS-Complete possibilities are explored. This may explain the apparent unthinking selection of H and Q for specialty engineering and environmental requirements, respectively. The intent is to identify all possible modeling artifacts with a letter as a source from which every conceivable requirement may be derived. Model letters for UML artifacts have been included in the set and will be introduced later.

This view provides clear traceability between the models from which the requirements were derived and the product entities to which they were allocated for all of the requirements, not just the performance requirements. What the author calls lateral traceability is captured in the database implementing the RAS-Complete. It is also a simple matter to link the rows in the matrix in a database to the corresponding verification requirements as well as the tasks to which they are allocated and their corresponding plans, procedures, and reports. Vertical traceability is, of course, simply a matter of relating the unique RIDs from pairs of requirements in specification parent-child relationships identified by their PID.
2.1.11 Traditional Structured Analysis Summary

In summary, a system is defined by identifying its functionality starting with the need (F), allocating that functionality to entities that become part of the system architecture (A). These entities that form the system architecture are selected by determining the performance requirements that the system must satisfy to meet the top-level customer’s need. The pairs of functions and product entity allocations pre-determine how the entities will have to relate to each other through interfaces (I) between the product entities. The environmental elements (E) are defined at the system level in terms of the spaces within which the system must function and the corresponding characteristics of those spaces drawn from appropriate environmental standards covering those spaces. As depicted in Figure 18, the traditional structured analysis effort attempts to define the most cost effective solution such that in N cycles of the process axis of the physical system (generally cyclical in the interest of reuse of system elements) the relation P (process) maps the cross product of the power sets of architecture (A*), interface (I*), and environment (Q*) to the function set F such that F is covered.
For every process Pi, there exists a combination of architecture, interface relations, and environmental stresses such that some subset of the function set is covered or accomplished. The power sets of these entities include all of the possible subsets of these entities within their own set thus the power set of A includes every useful combination of product entities relative to every process step. Useless subsets are also included in the power set as well, of course. It is important that the functions be covered in the correct order determined by the sequence of the processes linked together in the process axis. If all of the functions are satisfied in N revolutions of the process axis as planned, then we may say that the system is consistent relative to the use of its product entities, interfaces, and environmental stresses. If there are product entities that are not used in the process or some that are needed but not available we may not have the optimum product entity structure.

This whole process happens in practice somewhat backwards in that for an unprecedented system, one begins the development process only knowing the ultimate function, the need, and must expand everything from that one perspective.

2.1.12 SDD Content and Format

When used to support the application of traditional structured analysis on a program, the preferred SDD format consists of a main body and seven appendices, each providing a capture point for the work products of one of the several fundamental analytical system requirements analysis process areas. Figure 19 shows how the document is structured. A series of seven interactive system analysis activities feed the development of the appended data explained in subordinate paragraphs. The appended data then becomes the basis for lower tier analysis that produces content for the lower tier specifications and adds to the appended data.
2.1.12.1 Document Main Body

The main body simply contains a table of contents, list of illustrations, and list of tables for the document plus it should provide text explaining the capture of work products in the seven work areas during system and lower tier analyses. The body should also explain that the SDD couples the structured analysis work and its work products to specification content as guided by the selected specification standard templates.

2.1.12.2 Appendix A, Functional Analysis

This appendix captures the functional flow diagram starting with the identification of the system need and the life cycle flow diagram. The Use System Function is initially decomposed progressively to expose more details about the user need. For each block in the functional flow diagram, there should be one line in the function dictionary also contained in Appendix A.

2.1.12.3 Appendix B, System Environment Analysis

The environment consists of several subsets of stresses that are applied to the system. This appendix identifies and characterizes them. Timelines capture critical timing requirements. The spaces within which elements of the system must function are identified and the corresponding environmental stresses defined in terms of standards that describe those spaces. Service use profile analysis is applied to uncover end item environmental requirements. Finally, zoning of end items exposes component environmental requirements.
2.1.12.4 Appendix C, System Product Entity Analysis

The system product entities result from the allocation of functionality to things. As these pairs are defined on the function-product entity matrix, they must be entered into the product entity structure diagram. This work should be accomplished by a team of people knowledgeable in system, hardware, and software engineering, manufacturing engineering, verification engineering, logistics, material and procurement, and logistics in order to evolve an optimum product entity structure which will be universally respected on the program. This product entity structure is also the basis for the specification tree. Each item on the tree must have a responsible agent identified, a template selected, and a release date established. This structure should also be the basis for any IPPT established on the program. It is also the basis for the WBS so the SOW and IMP align perfectly with the teaming structure applied on the program.

2.1.12.5 Appendix D, System Interface Analysis

Interfaces are identified by pair-wise evaluation of function allocations to product entities using an n-square diagram. This appendix identifies all interface needs internal to the system as well as externally to the cooperative systems identified in Appendix B.

2.1.12.6 Appendix E, Specialty Engineering Definition Analysis

Appendix E provides a space in which system engineers can capture their work directed at identifying the specialty engineering disciplines that will have to accomplish work on the various entities in the system product entity structure to define the appropriate requirements and subsequently the needed analyses to confirm that those requirements are being satisfied. A specialty engineering scoping matrix is used to report the results of that analysis.

2.1.12.7 Appendix F, System Process Analysis

Appendix F captures the results of a physical process analysis in the form of a process flow diagram. This is used by logistics engineers to drive out requirements related to training, support equipment, maintenance procedures (tech data content), and spares consumption. It is also needed to complete the environmental use profile study reported upon in Appendix B that drives environmental requirements for end items.

2.1.12.8 Appendix G, Requirements Analysis Sheet

The exposed functions are listed in the Requirements Analysis Sheet (RAS) contained in this appendix. Related performance requirements are defined and allocated to a product entity. These performance requirements have to have a paragraph number assigned, title identified, and they can be outputted into a specification following a particular template. That part of the work can be done inside a requirements database system. Ideally, all of this work would take place within a requirements database tool but some organizations may find it preferable for their purposes to do the traditional structured analysis work using pencil and paper followed by capture of the resulting requirements in a word processor or a computer database tool from which specifications can be generated.
In keeping with the Integrated RAS idea advanced in Paragraph 2.1.10, the RAS is not restricted to performance requirements. Specialty engineering, interface, and environmental requirements can also be included so that every requirement appearing in every specification on a program will transition from the analytical model from which it was derived into a specification via the requirements analysis sheet.

2.1.13 Team Activity During Requirements Work

The PIT is responsible for accomplishing all requirements analysis work focused on developing the system specification and the immediately subordinate specifications that will be the top-level specifications for the top level IPPT. In general, this analysis work will be accomplished using traditional structured analysis following the pattern described in this section. PIT initiates the analysis capturing the work products in the SDD thus initiating that document. Requirements derived from the modeling work are entered into the database application initiating the requirements capture. Any customer requirements documentation is also entered into the database and traceability established between these requirements and those developed from modeling efforts that appear in the system specification. Traceability is continued down to the subordinate specifications to be handed over to the top-level teams when formed.

The PIT identifies all system external interfaces and defines them in the system specification or the beginning of an interface control document. Internal interfaces are also identified and defined for the first tier entities below the system level and the next lower tier in order to complete the internal interface definition for the first tier. All requirements are entered into the requirements database application and traceability entered.

The PIT develops a specialty engineering scoping matrix and maps the needed disciplines to the product entities identified and coordinates the indicated domain experts to derive requirements for the system and top-level end item specifications. The system level environmental requirements are derived from system spaces identified and mapped to corresponding tailored standards.

With the top-level specifications completed, the PIT can bring the top-level IPPT aboard. As those teams assemble and become familiar with their specification and the program planning data prepared by the proposal team, they continue the requirements analysis process relative to the functionality of their product entity. The primary role of the PIT switches to integration and optimization across the IPPT. The teams enter requirements data into the requirements database and maintain traceability. As this process continues, it may become apparent that lower tier teams are required in which case the parent team takes over system responsibilities for them. The parent team in all cases must develop the top-level specification for any new team. Also, at some point, a team will identify an allocation of functionality to computer software and the continued analysis of that entity should switch to UML.
2.2  UML

2.2.1  Entry Analysis and Overview

While it is encouraged that the enterprise apply TSA today as the entry modeling technique, it is entirely possible to initially enter the problem space analysis for a system that will be implemented primarily in software with UML rather than TSA or SysML. The suggested process starts at the top with the problem expressed in the system need and illustrated in a context diagram, borrowed from modern structured analysis, like that shown in Figure 20. The context diagram expresses relationships between the system, represented by the bubble, and terminators, representing outside entities deriving benefits from the system and supplying things needed by the system to function, generally information in a computer software system but more general in a system that will be implemented using a collection of technologies. Figure 20 is an alternative to the traditional depiction of the general system as a block labeled System interacting with a block named Environment.

![Figure 20  Context Diagram](image)

While UML could be the entry modeling approach at the system level following the approach discussed in this section, the discussion to follow is based on the assumption that the problem space will be entered using TSA with a recognition at some point of a need to switch to UML as entities are identified that must be developed in software. Figure 21 illustrates a process for applying UML starting at whatever level in the system the program chooses to start applying it. Generally, this will be some level below the system level based on allocation of higher tier functionality to a software entity.

In a given system, the initial analysis may have identified one or more entities that will be implemented in computer software so for each of these separate entities one should build a context diagram. It should be noted that the context diagram was popularized in modern structured analysis and was not adopted by UML but it is a useful artifact with which to identify the inside-outside relationship between the software entity and the entities external to that software with which it must interact. The context diagram is offered as an intermediary view that will lend some discipline and order to the identification of use cases. We will identify one or more use cases for each terminator and perform a dynamic analysis on each of those use cases.
For each terminator in Figure 20, identify one or more use cases through which the intended functionality will be accomplished identifying the actors deriving benefits from the system to be created. One use case may expand into several extended or included use cases to cover a more complex situation. You will note that the opening gambit has been arranged to provide structure in the identification of needed use cases. Next, for each use case, build one or more scenarios dealing with how that use case interacts with the system. These scenarios can be in text form, a list of events, or some kind of diagrammatic treatment.

Then, express each scenario in the form of an activity and/or sequence diagram cast at the UML entry level initially. The activity diagram can be thought of a functional flow diagram similar to that used by software programmers many years ago or by system engineer in traditional structured analysis. It may be drawn vertically or horizontally as far as the author is concerned. Swim lanes may be overlaid upon the activity diagram each of which corresponds with the lower tier entities (classifiers in UML) that will be responsible for implementing activities within their swim lane. This is a key point in the analysis where the analyst must make lower tier product entity structure decisions that should lead to adding software entities to the product entity structure. Some analysts prefer to think of the two-dimensional artifacts in the diagram as states rather than activities or functions. Simultaneity is not permitted in normal state diagramming but UML permits it in activity diagrams to cover decisions and branching in a way similar to that applied in flow charting.

Alternatively, the analyst can use a sequence diagram to open the exposure of the details of a use case scenario. The entities (classifiers) through which the functionality and behavior are explored are identified on what are called the life-lines drawn as dashed lines below selected physical or
logical classifiers. These lifelines correspond with the swim lanes on the matching activity diagram. Directed line segments connect these life-lines to show passage of messages and relationships between the classifiers. As in the activity diagramming, these life-line decisions may identify entities that already have been identified or they may involve entities not previously identified. In the latter case, the PIT must concur in the model extension and add the new entities to the product entity structure.

Each of these diagrams (activity and sequence) identify a lower tier (white box) view of what entities will be needed to accomplish the exposed functionality and behavior, what will have to happen in the system in order to achieve the intended goals of the use case signified in its name, and offer a detailed view of the order in which these things will occur.

The analyst can allocate top-level functions (activities) to specific entities and arrange the blocks of the activity diagram in corresponding swim lanes linked to these entities. These swim lanes will correspond to nodes, or even higher-level entities, at the system level but in any case we can refer to them in general as physical or logical classifiers within UML. If appropriate, analyze each of these classifiers from a dynamic perspective using some combination of sequence, communication, activity, and/or state diagram. The communication diagram shows the same information as a sequence diagram with an emphasis on the entities rather than the relationships between those entities. A state diagram is useful where there exists some finite number of possible conditions within which the entity can exist and there appear to be understandable rules for the entity to change from one condition (state) to another.

Identify requirements derived from these artifacts and capture them in a program-wide RAS linked to the modeling artifact (MID) that encouraged their identification and the physical or logical classifier, referred to more generally by a product entity ID (PID) in the RAS, that will be responsible for responding to that requirement and in the specification for which it should reside.

2.2.2 The Connection Between Modeling Artifacts, Specification Content, and Product Entities

Figure 22 suggests a hierarchical relationship between the elements of the UML analysis and offers a way of assigning MID. The capabilities in the specification format (template Paragraph 3.2) coordinate with terminators, use cases, extended use cases, and/or scenarios. Figure 22 only shows one terminator expanded but the intent is that for any top level software classifier AX (highest tier SW entities) entered into the product entity structure, one or more terminators would be identified and expanded as shown in the one case shown in Figure 22. The software top-level software classifier, AX, is, of course, a member of the product entity structure (PID) where X is a string of base sixty or decimal delimited base ten numerals. The suggested UML MID stream is identified first with a unique UXh MID (with h = e{1, 2, 3} in the example shown in Figure 22) for each terminator.

The terminator MID can be further decomposed using the MID UXhijk pattern composed from a set of use cases, possible extended use cases, and scenarios for each terminator h. These MID are the entries to place in the RAS for software requirements derived from these artifacts. In general, capabilities will be derived from these artifacts and the requirements subordinate to them will be derived form the dynamic modeling artifacts UXhijk1 through UXhijk4.
So finally, the requirements for each capability flow out of applying the subordinate UML dynamic modeling artifacts. As in TSA with lower tier product entities identified from lower tier function allocation, the lower tier software product entities flow from the sequence (life-lines) and/or activity (swim lanes) diagramming. This is a particularly satisfying coordination between lower tier entities being exposed through functional analysis in TSA and activity analysis in UML where both are using essentially the same model to identify lower tier entities.

We can continue to apply UML in the lower tiers as covered in Figure 21 treating each classifier as a system in accordance with the steps discussed above progressively identifying nodes, components (possibly in more than one layer), and classes. If the system level problem space was entered using UML as this process moves forward and downward, it will be decided that some classifiers will be developed as software and hardware entities, with the latter possibly splitting in lower tiers into hardware and software entities. The continuing analysis of hardware entities can be accomplished using traditional structured analysis or SysML model artifacts while the modeling of software entities continues primarily using UML model artifacts. In that it is
difficult for software to contain hardware, the use of TSA as the entry analysis probably makes more sense.

At the lower tier UML analyses where the physical and logical classifiers are classes and objects, the lines that flow between the classifiers on the corresponding sequence and communication diagrams coordinate with messages and influences applied to/from those classes in relation to external physical and logical classifiers (other classes generally at this level). This same effect is in operation whether the classifiers depicted are classes, components, or nodes. Each classifier must have identified for it one or more operations (services or functions) that it performs relative to an outside set of actors and one or more data entities it deals with internally to accomplish those operations. The data elements will flow into the classifier via the lines on the sequence and communications diagrams and data may be created or altered internally. Initially, the analyst may choose to first identify node, component, class responsibilities and subsequently as the analysis matures translate these responsibilities into operations and attributes.

In lower tier analyses, the assigned IPPT are responsible for identifying and defining needed interfaces below the level initially identified by the PIT. Where the two terminals of an interface touch only entities that are the responsibility of a single team, that team is clearly responsible for interface identification, definition, and integration. Where an IPPT is responsible for only one terminal of an interface, that team must cooperate with the team responsible for the other terminal to develop that interface. The integration agent in this case is the lowest common team. If there is only a single layer of teams under the PIT, the PIT is always the lowest common team. In general, it should be the team on the receiving end of an interface that first identifies the need for a new interface since interfaces should not be defined based on what is available but what is needed. If it is not obvious what team shall be responsible for the new interface, the PIT shall act as the integration agent until such time as the source is identified and then the lowest common team will take over that responsibility.

As IPPT identify lower tier entities during use case analysis, the PIT shall approve those additions and assemble them into the formal product entity structure. These entities may be physical or logical entities but eventually all of them must be identified as real product entities that will be developed. These final entities can be logical entities as in the case of computer software that will run on a particular hardware computer entity. Where it appears that lower tier entities will entail significant complexity, new IPPT may be formed by the PIT that will be subordinate to the appropriate existing IPPT. Those lower tier IPPT will take over the continuing analysis of use cases appropriate at that level and the immediately superior team will take on the role of the system engineer for the new team just as the PIT does for all top tier IPPT.

The use case analysis process employed by an IPPT or a collection of teams will necessarily be a collaborative process involving people from several different specialty disciplines. Each such team will have a leader whose responsibility it is to bring the analysis to a conclusion as scheduled. These teams will come into being, exist for a brief period of time, come to a conclusion, and pass from the scene with others replacing them. Once a use case analysis has been completed by a team, the work products will have been captured in modeling applications and integrated relative the existing work products. The modeling front will move down through the advancing product entity structure till the system is fully characterized.
Where personnel from other teams are required to accomplish work on another team's use case analysis, the owning team will cover the manhours of all personnel working the use case. Where all team members are physically collocated in the same facility, they may be depended upon to interact well through a traditional meeting in the common facility. Where at least some of the people required on a use case analysis effort are not collocated, it will be necessary to extend the meeting place geographically through the use of a product such as webex where people from several different geographical locations can cooperate in the development of a set of information.

Leaders of use case analysis teams are responsible for the prompt completion of team activity with good results but in so doing they will be well served to identify the most effective collaborative engineer on the team to lead collaborative team activity in the form of meetings especially where those meetings entail the use of distance integration aids like webex. The collaborative discussions in meetings need not necessarily be led by the team leader.

As the team completes its modeling and requirements work, the results should be reviewed by the parent team and, if approved, the team should be empowered to proceed with design at a level appropriate to the team responsibilities. The team must continue any responsibilities it may have for integration and optimization and leadership of subordinate teams that may still be involved in modeling work. This design work will entail some combination of hardware and software design development.

2.2.3 Dynamic Modeling Artifacts Explained

The use case diagram is considered a dynamic modeling artifact also but it is treated here as a transition medium between classifiers and the dynamic model set. The remaining dynamic models are implemented in four diagrams from which we may select any subset including all of them, any one of them, any pair, or any trio for a particular scenario analysis. It is not necessary to use them all for any particular analysis. Use those that make it possible to understand the problem space and properly characterize it in requirements. Some very large programs have done much of the analysis with only use case and sequence diagrams. The more complex the problem space and the more intimately the parts of the evolving system interact, the more different views of problem space that will be useful.

The first two kinds of diagrams covered, sequence and communications are both modeling the same relationships and collectively referred to as interaction diagrams. The sequence diagram emphasizes the time ordering of messages and the communication diagram emphasizes the organization of the objects that participate in the interaction. The second two are forms of state diagrams in the mind of many analysts.

2.2.3.1 Sequence Diagram

Some programs apply only the sequence diagram to explore the dynamic behavior of use cases and this may be adequate on relatively simple problems. In this tutorial we are assuming that the analyst employs the sequence diagram as the initial dynamic model, though an alternate approach would be to use the activity diagram for that purpose, but uses at least some of the
other three models to explore the use case more thoroughly. The sequence diagram example in Figure 23 illustrates the fundamentals showing two classifiers that comprise the classifier AX that has previously been analyzed. Here we conclude that AX must consist of AX1 and AX2 and that in order to accomplish the behavior defined for AX these lower tier classifiers must interact with an outside entity called an actor which will derive some kind of benefits from the relationship. The two subordinate classifiers will provide certain operations that are not clearly defined on this diagram. In the process of doing so, they will exchange messages in a certain order with time running down the page.

Each classifier including the actor has a lifeline in the form of a dashed line running down the page. A block is overlaid on this dashed line to indicate the active period(s) of classifier. Between these classifier lifelines we see messages being passed from one classifier to another. The names of these classifiers are formed of one or more words concatenated together without spaces and all but the first word capitalized. After the message name one can insert an argument list parenthetically.

The model permits the creation of classifiers and when they have performed their activity they can be killed. These features are not illustrated in Figure 23. The kinds of messages identified are: (1) a call invokes an operation on a classifier on the arrow end of the message, (2) a return message returns a value to the caller (dashed line used), (3) a send message sends a signal to a classifier, (4) a create message creates a classifier, and (5) a destroy message destroys a classifier. Message two and four could be an example of messages types 1 and 2. Message three is an example of message type 1 where the classifier is making the call upon itself.

![Sequence Diagram Example](image)

**Figure 23**  Sequence Diagram Example

After a classifier sends a signal to another classifier the sending classifier returns continues its own execution. The target classifier independently decides what to do about it. A common reaction would be to trigger a state machine causing the target classifier to execute actions and change state.
2.2.3.2 Communication Diagram

In some cases we are primarily interested in the time ordered sequence of messages between classifiers but in other situations we may be more interested in the organization of the classifiers and a communication diagram can offer better results even thought the sequence and communication diagrams are semantically equivalent. Figure 24 illustrates a communication diagram reflecting the same situation as Figure 23. The classifiers are joined by lines corresponding to the relationships between them and messages are related to these line each in terms of message name, direction, and the relative timing of the message.

![Communication Diagram Example](image)

2.2.3.3 Activity Diagram

An activity diagram can be used to express the things that one or more classifiers must accomplish. It is weak in terms of absolute timing of accomplishing those things but strong in expressing the relative order of those things. As shown in Figure 25, activities are illustrated by rounded corner boxes and they are connected into a sequence by directed line segments. In addition to the activities, we also will wish to show simultaneity through the use of forking and joining structures and alternative paths using branch and merge structures. The guard expressions give information about the conditions that correspond to movement through one branch or another.

This is the functional flow diagram of UML though many analysts prefer to think of the blocks as states rather than functions. The author prefers not to consider them states because it is in conflict with the notion that an entity must be in only one state at a time.

The diagram can be built with swim lanes, or not, that relate to classifiers, the same classifiers identified on sequence diagrams using the lifelines. It is through these two devices that we can allocate software functionality to classifiers. As we do so we determine the next lower tier product entity structure and should offer up newly identified entities to the PIT for inclusion in the product entity structure.
2.2.3.4 State Diagram

An interaction diagram (sequence or communication) models the relationships between a collection of classifiers while a state machine models the behavior of a single classifier. The classifier in this case can be the whole system or a classifier at any level of abstraction. The state machine models the possible condition that a classifier can exist in and the transition of that classifier from one condition or state to another based on a stimulus that might be a signal from another classifier, the passage of time, receipt of a call message invoking an operation, or a change in some condition.

A state is drawn on a state diagram, as shown in Figure 26 showing a state machine for temperature control, as a rounded corner box with the name of the state written inside. Generally the diagram must have an entry and final state symbols though it is possible that once entered the state machine may continue forever. In some cases the intent may be for the machine to continue forever, as in a traffic light system, the reality is that such a system may have to be shut down for maintenance and does need a final state. Transitions between states triggered by events in the life of the classifier being modeled are shown diagrammatically as directed line segments between a pair of state and named in a way to convey how that transition is triggered. It is understood that the machine must be in only one state at a time and that only a single transition is possible at one time. Transitions are generally thought of as taking place instantaneously.
While UML does not prescribe it, a pair of dictionaries can be helpful in clearly stating the intended operation of a state diagram. One dictionary lists the states and defines them with precision while the other lists and defines the transitions. You will note that the transitions in Figure 26 have not been named uniquely but should be in the general case so that each can one can differentiate between them in such a listing.

2.2.4  Static Entity Analysis

In early object oriented analysis (OOA) as prescribed by Grady Booch, Peter Coad teamed with Edward Yourdon, James Rumbaugh, et. al., and many other practitioners, the proper way to enter problem space was using the static view with objects. Then they encouraged the analysis of the objects from a dynamic perspective with data flow diagrams for functionality and state diagrams for behavior. This approach is possible with UML using the foursome of dynamic modeling artifacts discussed in the prior section and it can be effective when developing a largely precedent system that can be observed in the real World like a new payroll system. The author believes that largely unprecedented problems are best attacked using Sullivan's encouragement of form follows function leading with the dynamics views and identifying the static entities that populate the product entity structure from a software perspective.

UML identifies three levels of static entities but they are all product entities and while drawn on modeling diagrams using different images, they are essentially the same at different levels of abstraction. All are simply illustrated on the system product entity diagram illustrated in Figure 14 as blocks. Figure 27 illustrates the three static entities collectively referred to as classifiers in this tutorial.
In this tutorial the case is made for first identifying the nodes which are entities that will be associated with run time software. They are higher-tier entities. Like classes and components, they have associated attributes and operations. They interface with each other and possess lower tier interfaces between components and classes that comprise them. The nodes are identified in the dynamic analysis of the top-level software entity by identifying sequence diagram lifelines and activity diagram swim lanes. We then analyze these nodes from a dynamic perspective and identify components in the same fashion.

The alternative approach first identifies classes corresponding to observable entities in the problem space which are then dynamically analyzed leading to an understanding about how best to package these entities based on collecting the classes with the most intense interface relationships together as components. Just as in the use of interface analysis in TSA to validate the product entity decisions in functional analysis by observing possible unintended interface intensities, we can in UML re-consider the particular swim lanes and lifelines we selected in the dynamic analysis.

In this section, the intent is to explain what the UML static entities are and how they are used on diagrams. We will use classes in order to do so with the understanding that nodes and components are but higher tier classes. First we will describe a general class then explore structural relationships between these classes and finally we will cover the messages that are passed between them.

2.2.4.1 The Class

A class is illustrated as a box as shown in Figure 27c. The name of the class is placed in the top portion of the box, attributes are listed in the middle portion of the box, and operations are listed in the lower portion of the box. A forth box can be included below the operations in which we inscribe class responsibilities in free-form text. A responsibility is a contract or an obligation of a class. You may find it useful to begin the analysis of classes this way translating these into attributes and operations that best fulfill the class’s responsibility as the model is refined.
2.2.4.2 Class Relationships

Figure 28 illustrates the structural relationships recognized between classes. A class is said to be dependent on another if it depends on that other class for information or services. A class can be linked hierarchically to another through a generalization. Class associations can have the three adornments illustrated in Figure 29.

- **Association Name**
  - NameOne \( \xrightarrow{x..y} \) Association name \( \xrightarrow{\text{role}} \) NameTwo

- **Association Role**
  - The face that the class at the far end of an association presents to the class at near end of the association. Role names called end names.

- **Association Multiplicity**
  - Tells how many objects may be connected across an association instance. Given by a range of numbers.

- **Association Aggregation**
  - Expresses a whole-part relationship between to associated classes.

Figure 28 Structural Relationships

Figure 29 Association Adornments
2.2.4.3 Messages

Classes can also be related through the five kinds of messages discussed in Paragraph 2.2.3.1. Such a message can convey to a class a variable argument (value) that is needed in a class operation or it can convey a signal that causes the class state machine to transition to a new state for example. The messages that must be passed between classes are understood in the context of the sequence diagram under the assumption that the dynamic analysis is accomplished prior to the static analysis.

2.2.5 Related Analyses

2.2.5.1 Specialty Engineering

The specialty engineering matrix discussed in paragraph 2.1.9 can be used in software as well as hardware to identify all product entities for which specialty engineering requirements must be developed. This includes, for example, safety, reliability, and security. The software interface requirements fall out of the sequence and communication diagram analyses and flow into the specification template offered in Figure 27.

2.2.5.2 Environmental Requirements

Software environmental requirements are somewhat different from the hardware and system environmental requirements that tend to be dominated by the natural environmental factors. The software being an intellectual entity rather than a physical one, is shielded from the natural environmental stresses. True, the software operating within a machine that can suffer adverse environmental stresses can as a result fail, but this is a secondary effect. Reasonable software environmental relationships include any language restrictions and the computer architecture upon which it must run, for example.

2.2.6 Specification Structure

The specification outline offered in Figure 5 can also be applied to software entities where the capabilities are linked to either the terminators, use cases, extended use cases, or scenarios and the subordinate requirements listed under each capability are drawn from the corresponding dynamic diagramming (activity, sequence, communication, and state diagramming) work. Thus the requirements can be clearly shown to trace to modeling artifacts just as the performance requirements in hardware can be shown to flow so clearly from functions. Figure 30 offers an outline for a software requirements specification (SRS) using an edited template from EIA J STD 016 to integrate the supporting modeling work into the specification. Note the similarity to the outline in Figure 5 for a system or hardware item performance specification.
<table>
<thead>
<tr>
<th>PARA</th>
<th>TITLE</th>
<th>MODEL</th>
<th>DEPT</th>
<th>APP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>SCOPE</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>2.</td>
<td>APPLICABLE DOCUMENTS</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.</td>
<td>REQUIREMENTS</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1</td>
<td>Required states and modes</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.1</td>
<td>Classifier context diagram</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.2</td>
<td>Use case analysis</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.2.h</td>
<td>Terminator h</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.2.h.i</td>
<td>Terminator h, use case i</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.2.h.i.j</td>
<td>Terminator h, use case i, extended use case j</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.2.h.i.j.k</td>
<td>Terminator h, use case i, extended use case j, scenario k</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.3</td>
<td>Dynamic Analysis</td>
<td>DID</td>
<td>2100</td>
<td>D</td>
</tr>
<tr>
<td>3.1.3.h</td>
<td>Terminator h dynamic analysis</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.3.h.i</td>
<td>Use case hi dynamic analysis</td>
<td>UML</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.3.h.i.j</td>
<td>Extended use case hij dynamic analysis</td>
<td>UML</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.3.h.i.j.k</td>
<td>Scenario hij dynamic analysis</td>
<td>UML</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.3.h.i.j.k.1</td>
<td>Sequence diagram hijk1 analysis</td>
<td>UML</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.3.h.i.j.k.2</td>
<td>Communication diagram hijk2 analysis</td>
<td>UML</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.3.h.i.j.k.3</td>
<td>Activity diagram hijk3 analysis</td>
<td>UML</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.3.h.i.j.k.4</td>
<td>State diagram hijk4 analysis</td>
<td>UML</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.4</td>
<td>Lower tier classifier identification</td>
<td>UML</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.2</td>
<td>Entity capability requirements</td>
<td>UML</td>
<td>2100</td>
<td>A</td>
</tr>
<tr>
<td>3.2.m</td>
<td>Capability m</td>
<td>UML</td>
<td>2100</td>
<td>A</td>
</tr>
<tr>
<td>3.2.m.n</td>
<td>Capability m, requirement n</td>
<td>UML</td>
<td>2100</td>
<td>A</td>
</tr>
<tr>
<td>3.3</td>
<td>Interface requirements</td>
<td>UML</td>
<td>2100</td>
<td>D</td>
</tr>
<tr>
<td>3.3.1</td>
<td>External interface requirements</td>
<td>UML</td>
<td>2100</td>
<td>D</td>
</tr>
<tr>
<td>3.3.1.m</td>
<td>External interface m</td>
<td>UML</td>
<td>2100</td>
<td>D</td>
</tr>
<tr>
<td>3.3.1.m.n</td>
<td>External interface m, requirement n</td>
<td>UML</td>
<td>2100</td>
<td>D</td>
</tr>
<tr>
<td>3.3.2</td>
<td>Internal interface requirements</td>
<td>UML</td>
<td>2100</td>
<td>D</td>
</tr>
<tr>
<td>3.3.2.m</td>
<td>Internal interface m</td>
<td>UML</td>
<td>2100</td>
<td>D</td>
</tr>
<tr>
<td>3.3.2.m.n</td>
<td>Internal Interface m, requirement n</td>
<td>UML</td>
<td>2100</td>
<td>D</td>
</tr>
<tr>
<td>3.4</td>
<td>Specialty engineering requirements</td>
<td>DID</td>
<td>2100</td>
<td>E</td>
</tr>
<tr>
<td>3.4.m</td>
<td>Specialty Engineering Discipline m</td>
<td>Specialty Scoping</td>
<td>E</td>
<td></td>
</tr>
<tr>
<td>3.4.m.n</td>
<td>Specialty Engineering Discipline m, Requirement n</td>
<td>Specialty Scoping</td>
<td>E</td>
<td></td>
</tr>
<tr>
<td>3.5</td>
<td>Environmental conditions</td>
<td>3-Layered Env Model</td>
<td>2100</td>
<td>B</td>
</tr>
<tr>
<td>3.6</td>
<td>Precedence and criticality of requirements</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>4.</td>
<td>VERIFICATION</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>PACKAGING</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>6.</td>
<td>NOTES</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>6.1</td>
<td>Requirements traceability</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>6.1.1</td>
<td>Inter-specification traceability</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>6.1.2</td>
<td>Verification traceability</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>6.1.3</td>
<td>Modeling traceability</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>6.1.4</td>
<td>Section 2 traceability</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>6.1.5</td>
<td>Programmatic traceability</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>6.2</td>
<td>Glossary</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>6.3</td>
<td>Specification maturity tracking</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
</tbody>
</table>

Figure 30  Software Requirements Specification (SRS) Template
Refer to Exhibit B of this student manual for a JOG System Engineering data item description (DID) that shows how the subparagraphs of paragraph 3.1 relate to the subparagraphs of paragraph 3.2. Paragraph 3.1 essentially provides an opportunity to capture the complete UML analysis for the classifier covered in the specification. This can be done by actually including the diagrams discussed in prior paragraphs in this text or by referencing them in a computer application within which the modeling is done and derived requirements captured or the diagrams can be completed manually (or with a computer graphics application like Microsoft Visio or Powerpoint) and contained in the appendices of the system definition document (SDD).

The reader will note that in Figure 19 there is a second plane labeled UML for the case where the program chooses to capture the UML analysis work products in the SDD. On a program that is dealing only with a UML analysis for a software product the Appendix structure shown in Table 1 column A could be used. If the program must deal with both TSA and UML, the software appendices could be simply added to those required for TSA and lettered G through N with the TSA Appendix G (the RAS that should also contain the software requirements derived from UML artifacts using the MID pattern illustrated in Figure 22) becoming a common RAS in Appendix O. The latter pattern is captured in column B. In this combined case, the classifier diagramming can remain in Appendix N but Appendix C should capture the aggregate product entity structure including hardware and software entities.

<table>
<thead>
<tr>
<th>Table 1 Independent and Combined SDD Appendices</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
</tr>
<tr>
<td>----</td>
</tr>
<tr>
<td>A</td>
</tr>
<tr>
<td>B</td>
</tr>
<tr>
<td>C</td>
</tr>
<tr>
<td>D</td>
</tr>
<tr>
<td>E</td>
</tr>
<tr>
<td>F</td>
</tr>
<tr>
<td>G</td>
</tr>
<tr>
<td>H</td>
</tr>
<tr>
<td>I</td>
</tr>
</tbody>
</table>

The reader should note that in software using UML it is possible to relate the modeling machinery very clearly with the classifiers about which specifications are to be written. It is not quite so easy in hardware using TSA because all of the performance requirements derived from a particular function may be allocated to the same entity. It might be possible to force fit the functions into a closer alignment with the product entity structure by restarting the functional analysis for each entity identified but the author does not think this would be particularly helpful.

2.2.7 Software Requirements Close-Out

When a set of classes that collectively form a component have been thoroughly characterized including the sets of requirements associated with those classes and dynamic relationships and the related requirements have been captured, reviewed, and approved, the responsible team can
start to develop the computer code that will implement the component. This process can begin across the lower tier of the components and continue to next assemble into computer code for the nodes. This process may take place in a single string or simultaneously following the same pattern in multiple paths where the software is distributed to some extent. There is, of course, a continuing need for integration and optimization across the development being accomplished by different teams that focus primarily on interfaces just as in hardware development.

The PIT and program manager will reach a conclusion for the program with possible local differences whether to pursue software design and coding in a top-down or bottom-up fashion. In either case, any code developed must be tested at each level of build. This may require the development of special stub (top-down) and/or drivers (bottom-up) to permit testing of completed portions before higher or lower tier software has been completed.

The classifiers identified in the UML analysis should be entered into the product entity structure by the PIT so that a consolidated view of the overall system matures. Figure 31 shows the product entity structure evolving based on feed from all modeling activities being applied on the program under the watchful integrating eye of the PIT.

![Evolving Product Entity Structure](image)

The UML analysis process encouraged does involve a partitioning of the whole problem space into parts hierarchically arranged and assigned to teams. These teams, in hardware and software, can generally be depended upon to integrate and optimize at the level for which they are responsible rather than the next higher level. Therefore the PIT (and superior IPPT) must integrate and optimize across the best efforts of the teams. The PIT should also consider special integration studies across the grain of the system based on overarching functions such as shutdown and activation as well as others that may be suggested through system level states and modes analysis. These efforts should be led by the PIT and participated in by people from a wide range of teams.

2.3 Opening the Analysis With DoDAF

DoDAF was developed to support modeling of the complex information systems that DoD has been assembling in recent years to interconnect their many sensor and weapons systems to form
an effective military capability often referred to as sensor-to-shooter connectivity. It does not provide a complete modeling set primarily because of the absence of a model artifact for the physical product entities. The product entity view can be provided by augmenting the DoDAF model set with the TSA product entity diagram, however, from the aggregate model set this description assembles. Physical products identified in the DoDAF process can be inserted into the TSA product entity diagram just as they can be using any of the model sets and as suggested in Figure 31.

The DoDAF problem space entry is similar to the UML entry and in fact can use some UML artifacts to do so. DoD does not prescribe any particular set of modeling artifacts for the several views. In this paper we will use a combination of UML, IDEF, and TSA artifacts. DoDAF employs four modeling views: (1) two all view products that offer a textual overview of the system (as in what was once called a mission need statement) and an overall glossary explaining all terms used; (2) two technical view products that define standards that must be respected on the program and the evolution of those standards over time; (3) nine operational view products that capture how the user views their needs; and (4) thirteen system view products that offer the engineering perspective on the needed system. Figure 32 illustrates a recommended DoDAF development sequence. The simulation work could better be shown overarching both the analysis and design work interacting with and serving both. Note that the two all and two technical views would be a good addition to any modeling apparatus.

Figure 32  DoDAF Development Sequence

The technical and all views work products should be developed initially by the user and acquisition agents and evolved by the contractors selected to develop the system. The user should have developed the operational views in the process of evolving a set of documents used to refine their need. These documents include a Joint Capabilities Document (JCD that takes the
place of the old need statement cut very broadly. In the study process prescribed by DoD for a new system, the capability identified in the JCD may be cut up into n systems each of which should have developed for it an Initial Capabilities Document (ICD) that is the equivalent of the old mission needs statement for a given system to be procured. The ICD is then matured into a Capabilities Development Document (CDD) and eventually a Capabilities Production Document (CPD). In between these last two a contract would commonly be let resulting in the development of a system specification attached to the contract. DoDAF can be used to gain a systematic insight into the content needed in all of these documents.

2.4 Integrated Modeling

A program must make a decision about the modeling techniques it will apply as it builds the proposal. This may include some mix of TSA, UML, SysML, DoDAF, and IDEF-0. A program that is going to be primarily computer software or networked assets could enter the program with UML or DoDAF. If the product is a database, it could enter with IDEF-0 or UML. But, generally, modeling entry should involve the use of TSA or SysML at the system level because software, an intellectual entity, must run on hardware entities that provide the product with real substance. As noted earlier, at the time this was written SysML was not yet fully ready for use so TSA would be the author's preference now. But, an enterprise should continue to follow the development of SysML and work toward replacing TSA with SysML. In any case, there is a need to recognize that for some period of time there will be a need for a model that works well for systems and hardware and another model that works well for SW. For now, also, there is not one great computer application within which one can model HW and SW requirements work and permit easy cross model traceability and provide specification publication capability so it will be necessary to use two or three applications to cover the needed tool set.

Work can be accomplished well for systems and HW as covered under paragraph 2.1 and for SW using the approach covered under paragraph 2.2. When applying both, problems will tend to arise when transitions have to be made between these two approaches. It is not possible for SW to include HW but the opposite case is perfectly normal. So, the transitions will only be a problem as the analysis shifts from HW to SW moving from the use of TSA or SysML to UML. There are two concerns at this point: one in the models applied and the other in the computer applications employed.

The transition point will occur when the highest tier software entities are identified. There may, of course, be several of these transitions distributed about the expanding product entity structure. The program has the option of pooling all of the software into an integrated entity or permitting it to be distributed within multiple processors that may still all be under the responsibility of one team or distributed among teams with both hardware and software responsibilities. If we can solve one of these hardware-software handoffs we will have solved the general problem of requirements traceability across these gaps.

It should be clear that requirements traceability to models is assured in the approach covered in this tutorial because all of the requirements are to be derived from a model. Vertical or hierarchical requirements traceability is very simple in specialty engineering areas in hardware, software, and across the gap as described in this tutorial. The environmental requirements are
vitaly different between hardware and software and one can make a case that lower tier software environnental requirements should not have to respect traceability across the gap to higher tier hardware or system environmental requirements that are largely environmentally related. Precisely the same method of identifying hardware interface requirements can be used to identify software interfaces as well as hardware-software interfaces because we identify them between entities that appear in the joint product entity structure. So, if interface requirements traceability involves lower tier interface expansion requirements to higher tier interface requirements, traceability is assured. This leaves only the performance requirements a remaining problem from a vertical or hierarchical traceability perspective.

Given that the system entry analysis was accomplished in TSA using some form of functional analysis and the lower tier software analysis is going to be done using UML, there is a temptation to employ activity diagrams in UML to analyze software entities from a dynamic perspective because it is very similar to functional flow diagramming and might give us some interesting opportunities to link up hierarchical traceability. However, for a given software entity there may have been 10 performance requirements derived from 8 functions allocated to the software entity in question. There is no clear way to link up the activity analysis and requirements derived from it with the several functional analysis strings and the performance requirements derived from them that can easily be automated.

So, let us pursue another tack in an attempt to coordinate the traceability relative to the sequence oriented dynamic analysis approach described previously. If requirement R%1 is one of a set of requirements R% through R%10 where R%1 is derived from function F#1 of a set of functions F#1 through F#8 and requirement R%1 is allocated to product entity AX2 and it is decided that AX is going to be developed as a software entity, then one of the scenarios to be analyzed will be UXhijk. Assume that we accomplish the dynamic analysis using sequence diagram UXhijk1 from which we derive requirement R@1. What we are looking for is a way to establish hierarchical traceability between requirement R@1 and some requirement in the set R% through R%10. The X, %, and # characters are being used to designate base 60 strings in this discussion. We know that requirement R@1 must be traceable to one of the 10 performance requirements allocated to classifier AX2 and we can look at that list of requirements and select the one most closely related.

To make this selection more organized, we can form an x by y matrix, in this case a 10 by 12 matrix, and pair-wise compare the sets R@ and R%. In Figure 33 you can see this whole process taking place The 10 functionally derived requirements are captured in the RAS mapped to the set of functions F#1 through F#8 and allocated to product entity A&. Based on these requirements we build a context diagram for entity A& and analyze A& from the perspective of each of the three terminators shown. As an example Use Case U&3 is extended to three use cases and we build three scenarios one of which, U&3111 is analyzed from a dynamic perspective with some combination of sequence, communication, activity, and state diagrams. Requirements R@1 through R@12 are derived from these analyses and captured in the RAS (possibly linked to the RAS database from a UML modeling application).

There are 10 requirements (R%1 through R%10) to which the requirements in the set R@1 through R@12 will have to hierarchically trace. We can build a 10 x 12 matrix and pair-wise analyze the
relationships between the two sets of requirements, perhaps concluding that one of the matches is $R_{@1}$ traces to $R_{@2}$. All of the matches are marked in the requirements management database table for traceability relationships. There are no known databases that provide the traceability evaluation matrix so it may have to be accomplished as a pencil and paper aid. However we should be able to set the requirements management database filter for the two sets of requirements of interest aiding in the identification of the sets of interest for a particular case. In this example, there might be ten or more sets of requirements like $R@$ derived from ten or more dynamic analyses. In each case an $x$ by $y$ matrix would be needed to pair-wise analyze the traceability relationships. In any case, it should be clear that we can have good requirements to modeling traceability and even good hierarchical traceability across the gap between performance requirements.

Figure 33  Requirements Traceability Across the Gap

In both TSA and UML we have discussed a decomposition process that partitions the problem space into parts in which the analysis is accomplished. Whenever we partition any whole we have an obligation to integrate and optimize across the boundary conditions thus created. The PIT must accomplish this integration work relative to the top level IPPT and each IPPT with lower tier teams must accomplish this work relative to it's own immediately subordinate teams. Much of this integration work will take place at the interfaces ensuring that requirements on one end of an interface are compatible with those for the other terminal. Each team with subordinate teams, however, should also integrate across its immediately subordinate teams relative to the requirements derived at the subordinate team level relative to those at the parent team level. Part of this work can be accomplished by simply establishing the traceability between the requirements at the two levels. Another approach of value is to accomplish higher tier function effects across the lower tier team responsibilities. For example, one can inquire collaboratively into lower tier performance of higher tier functions like turning the system or entity on or off,
moving from one major mode to another, accomplishing some kind of transfer function, or physical separation or joining of two entities.

Another kind of traceability can also be used to stimulate integrating results. This was pointed out to the author by an engineer at Puget Sound Naval Base in Bremerton, WA. Given a requirement at level m, we can inquire if the intent of the requirement was fully implemented in the requirements for the n entities at level m+1 (downward). This kind of traceability inspection must await the development of the subordinate specifications, of course, as does all hierarchical traceability.

At one time, in the 1950s when software was a very young discipline, it happened that hardware and software analysis, to the extent that it was done, was accomplished using exactly the same model, flow charting as shown in Figure 34. Over time, probably encouraged by the ease with which flow charts could be outputted onto line printers using ASCII symbols, computer software people got into the habit of building flow charts in the vertical rather than the horizontal axis still used by system engineering in their functional flow diagrams. The activity diagrams of UML still reflect the vertical orientation but it is really of little significance which orientation is used. The absolutely fascinating approaching reality is that system and software people will rejoin the same house in the near future. As UML and SysML become more fully integrated as suggested in Figure 34, we will achieve a tremendous milestone of universal unified modeling capability.

As we pass through this door into a world of integrated modeling and supporting computer applications, we will find it a more reasonably affordable task to integrate across the hardware-software boundary than has been the case for many years. But then as now, integration takes place in the minds of the system engineers working on the program. These engineers must be
every vigilant for inconsistencies between information sets that signal that two different domains are not working from a common understanding of the problem and solution spaces.

Figure 35  The Approaching Merge

3  Requirements Management

3.1  Summary of Team Activity During Requirements Work

During initial product development work, the PIT will model the problem space using a predefined set of modeling methods selected from the list of enterprise-approved modeling methods and apply those methods to identify top-level system entities and interface relationships and their requirements. This level of system definition shall be completed before program level IPPT are initiated on the program and staffed. These top-level product entities are the basis for assignment of these teams. When a team is established, the leader shall be presented with the specification for the top-level entity for which the team shall be responsible, a design concept (in particular if it is HW or SW), a clear definition of all external interfaces, and the corresponding components of the WBS, SOW, IMP, and IMS. The team will be charged with the development of that item and all subordinate entities and interface relationships. In the process of so doing, the team may conclude that lower tier teams are required and must request that action of the PIT.

The work products from the IPPT will be loaded into the computer applications used on the program by PIT and checked for consistency. The PIT shall perform integration and optimization work on modeling work contributed by the IPPT fitting the work products into a coherent system analysis of the problem space. As part of this work where the product is to be implemented in SW, the responsible team will seek to identify all needed use case analyses and assign them for completion by people on the team if possible. Where this work must involve people from other
teams, the responsible team must request help from the PIT and a cross team analysis effort will be established. The responsible lead team must ensure that each analysis is complete with all needed supporting modeling work.

The PIT shall maintain the product entity structure, the interface relationships, and all requirements modeling and management assets. The requirements shall be retained in a relational database from which specifications may be printed to paper or computer screen and within which traceability may be maintained. This database shall be linked to one or more modeling applications used on the program. The modeling applications shall be used for the purpose of identifying the system entities, interfaces, and appropriate requirements in each case. All content of these applications shall be under the control of the PIT until such time that it is formally approved at which time it shall fall under configuration management control and shall not be changed without a formal approval as well. Responsibility for data entry can be distributed to IPPT or retained by PIT. Entry may be aided by special Microsoft Office applications making it unnecessary for personnel to develop and maintain computer application skills.

The PIT will seek to establish IPPT overlaid upon the product entity structure so as to minimize the interface relationships between the entities for which the teams are responsible. The purpose of this arrangement is to minimize the need for cross team coordination and staffing for use cases analyses.

The preferred lower tier HW development approach entails a continued application of TSA using the same modeling database application applied at the system level. The preferred SW product development approach entails PIT and IPPT application of unified modeling language (UML). In the near term traditional structured analysis (TSA) will be applied in combination with UML maturing to a combination of UML and SysML as the latter matures into a formally released model by the Object Modeling Group. TSA or SysML are to be used initially to identify system and top level product entities that will more often be hardware end items. As the analysis proceeds downward and identifies a need for computer software, the analysis should switch to UML.

3.2 Requirements Tools Base

Figure 36 illustrates the preferred tools environment for programs. A requirements management database is used to capture all of the requirements that will be published in specifications and those specifications may be published from this database. In Figure 36 this is referred to as a big dumb database with no slur intended for the makers of tools that do not include integrated modeling capabilities. This is a relatively simple relational database application that can contain text information in a tabular structure. Each table row corresponds to a unique requirement with data captured in table columns for the several fields needed. Additional relational tables may be required for vertical traceability, verification traceability, and lateral (to models from which the requirements are derived) traceability. The program may use available modeling applications for UML (such as Rational Rose) and TSA (such as CORE) and arrange for traceability between these applications and the management application in an application like DOORS.
Many enterprises find it difficult for engineers to maintain currency with a set of requirements database applications as well as other applications more directly related to their work. All or most of these engineers will, as a function of accomplishing their normal work, maintain proficiency with the three fundamental Microsoft Office applications (Word, Powerpoint, and Excel). Loader applications crafted from Microsoft Office applications may be used to permit all engineers to enter data into the requirements database suite without a need for the engineers to become intimately familiar with these applications.

The PIT must exercise integration and optimization control over the requirements application suite and will require some members who really understand the applications, how they work, and how their content is inter-related. The suit must be set up to permit passing control of approved content to configuration management while retaining control of all in-work data.

3.3 Recommended Specification Responsibility Pattern

In the author's view, a program should staff a program integration team (PIT) that should begin the requirements analysis process at the system level and develop the top level diagrams in the SDD. This work should continue as necessary to develop the content of the system specification and the specifications corresponding to the top-level teams. The structured analysis for each of these teams should be taken over by the corresponding IPPTs in each case until they have completed the content of the specifications that define the problem for any subordinate teams. If no subordinate teams have been identified then they would have to complete the analysis needed to develop all of the specifications subordinate to their top-level specification. This same pattern
carries down to the lowest level. Each team should act as the system agent for all of its lower tier teams and principal engineers. This starts at the PIT for the system and works its way down through the lower tier teams. The Program Manager and Chief Engineer/PIT Manager should review and approve the system specification and all top-level specifications. PIT should establish rules for review and approval of lower tier specifications created by the teams.

With different parties doing the structured analysis, it is necessary to apply process integration and the PIT should do that accepting data into the several appendices of the SDD, numbering the figures, and cross-checking the data submitted. At least one team will be involved in software development and if traditional structured analysis has been applied for the system level, then that or those teams responsible for software will want to switch to some form of software modeling such as UML. Regardless of the modeling methods applied, all of the requirements modeling artifacts should be captured in the SDD either in the paper appendices noted earlier or referenced in the database systems used. The integrated RAS should be implemented in the big dumb database of Figure 36 as simply the basic relational database table used to capture requirements.

3.4 Requirements Risk Management

The principal risks that appear during the requirements development work involve a sensed inability to satisfy the requirements. The risk may be motivated by the conclusion that insufficient financial, or schedule resources have been made available. The concern may be that the requirement simply cannot be satisfied with available technology. Finally, the concern may be motivated by the conclusion that the value is simply too hard to achieve with available skill and knowledge. It is not uncommon to partition all into the categories of cost, schedule, technology, and performance as a result.

3.4.1 Requirements Validation

EIA 632 identifies an activity called requirements validation where we make an effort to determine to what extent we can satisfy particular requirements. The simplest way of reaching this conclusion is to simply ask the person(s) responsible for accomplishing the related design work if they can satisfy the requirements. If there is a lack of confidence then we need to proceed further in our efforts to identify potential performance risks. As requirements are identified and written we should validate them at that time. In most cases, the conclusion will be that there is no problem. Should we conclude that either there is a problem or we are not certain that can satisfy the requirement the first alternative investigated should be to ask if the requirement can be changed making it more certain that it can be satisfied. If that is not possible, then we should choose a means to mitigate the risk through an appropriate analysis, development evaluation test, simulation, or demonstration. If we believe that the requirement is very important in the development effort and that it will require some time to reach a final conclusion, we may select the requirement for more intense management as a technical performance measurement (TPM).

Parameters are managed through TPM by placing them on a list and assigning each TPM to a specific engineer who is charged with closing the gap between the required value and the currently demonstrated capability. The parameter principal engineer must maintain two charts: (1) a parameter chart that tracks these two values over time annotated with notes citing important
events coinciding with significant changes in the relationship between the values and (2) an action plan stating what is going to be done, when, and to what anticipated effect.

3.4.2 Margins and Budgets

Every program manager will experience difficult problems each requiring a tough decision. These problems can be made less severe by ensuring that the program manager has the resources available. This outcome is encouraged where the values for the most difficult requirements are combined with margins such that there is an opportunity to award engineers with a very difficult design problem some slack. These margins come in three varieties: cost, schedule, and performance. Cost margin is commonly applied as a management reserve such that the program manager can award a team more cost to solve a problem. Similarly, if a team has a design problem that can be solved through award of schedule slack time, the design may be possible. The third kind of slack is requirements margins. In all of these cases, the margin is derived by invariably making the job more difficult. Cost margin is often made available by skimming the task estimates of 10-15%. Schedule slack is obtained by subtracting available time for tasks on the critical path. Risky requirements are made more difficult to achieve. For example, a weight margin may be realized by subtracting 5% from all weight figures. So the engineers will be challenged to accomplish their design with a weight value of required value - 5%. This is more difficult, clearly. The good news is that engineers will most often make these more demanding requirements preserving the margin values for the most difficult problems. When a very difficult weight problem appears, the manager can allocate some available margin. The margins invariably will be consumed before the design process is complete but there are ways of using available margins from requirements not of the same kind. For example, if an engineer has difficulty reaching his/her reliability figure after all of the reliability margin is gone, the manager can award some cost margin to permit the use of better parts or some mass margin permitting a heat sink that will reduce junction temperatures.

Requirements budgeting also has a risk reduction effect because it partitions available requirement values to the several designs at any one level of indenture. For example, given that it has been decided that 1500 watts of electrical power will be available from a source and there are 10 loads to be attached to this source, an engineer must partition this available power in a rational way between these several loads and integrate the results.

3.4.3 Risk Tracking

Risk is often measured using a dual axis criteria dealing with the probability that the concern will be realized and the degree of difficulty it will present if it does come to pass. This makes it a little difficult to track a single risk parameter over time and the way many people apply the dual axis system makes it difficult to accumulate a program metric that can be tracked over time. A variation on the safety hazard index described in MIL-STD-882 offers a way to measure risk with a single parameter that responds properly to characterize instantaneous values and a historical record for the program.

Figure 37 shows the risk matrix. Tables 2 and 3 corresponding provide the dictionaries explaining the values entered on the matrix axes. The intersections contain an index number that is simply the product of the axis numbers.
Table 2  Risk Probability of Occurrence Criteria

<table>
<thead>
<tr>
<th>CAT TITLE</th>
<th>P(O)</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nearly Certain</td>
<td>0.95-1.00</td>
<td>Will occur at least once during program</td>
</tr>
<tr>
<td>Probable</td>
<td>0.75-0.95</td>
<td>Will probably occur once during program</td>
</tr>
<tr>
<td>Possible</td>
<td>0.50-0.75</td>
<td>May occur during program</td>
</tr>
<tr>
<td>Unlikely</td>
<td>0.25-0.49</td>
<td>Will probably not occur during program</td>
</tr>
<tr>
<td>Nearly Impossible</td>
<td>0.00-0.24</td>
<td>Will not occur during program</td>
</tr>
</tbody>
</table>

Table 3  Risk Effects Criteria

<table>
<thead>
<tr>
<th>CAT TITLE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Catastrophic</td>
<td>Program in jeopardy of termination</td>
</tr>
<tr>
<td>Serious</td>
<td>Serious damage to program</td>
</tr>
<tr>
<td>Moderate</td>
<td>Problems cause program focus difficulties</td>
</tr>
<tr>
<td>Minor</td>
<td>Problems that can be easily be overcome</td>
</tr>
<tr>
<td>Null</td>
<td>No problem</td>
</tr>
</tbody>
</table>
If you were to compare this information with the MIL-STD-882 safety hazard matrix, you would find that the safety hazard matrix offered in the military standard uses letters for one of the two axes and that the highest hazards (risks) have the lowest indices. Our matrix in Figure 37 uses numbers on both axes and the index values are higher for more serious risks. Therefore, it is possible to apply the index values in a mathematical sense as a program metric. Given the 6 program risks listed in the program risk list shown in Table 4 with the indicated risk index values, the instantaneous program risk index is 97.

Table 4  Program Risk List

<table>
<thead>
<tr>
<th>RISK NBR</th>
<th>RISK TITLE</th>
<th>PROB</th>
<th>EFF</th>
<th>INDEX</th>
<th>TM</th>
<th>PRINCIPAL</th>
<th>SUSP</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Life Cycle Cost</td>
<td>4</td>
<td>4</td>
<td>16</td>
<td>1</td>
<td>Burns</td>
<td>02-10-05</td>
</tr>
<tr>
<td>5</td>
<td>Payload Capacity</td>
<td>4</td>
<td>5</td>
<td>20</td>
<td>1</td>
<td>Adams</td>
<td>03-08-05</td>
</tr>
<tr>
<td>7</td>
<td>Stoddard Supplier Risk</td>
<td>5</td>
<td>4</td>
<td>20</td>
<td>3</td>
<td>Thornton</td>
<td>04-20-05</td>
</tr>
<tr>
<td>12</td>
<td>Program Funding</td>
<td>4</td>
<td>4</td>
<td>16</td>
<td>0</td>
<td>Connolly</td>
<td>03-10-05</td>
</tr>
<tr>
<td>15</td>
<td>Computer Software Schedule</td>
<td>5</td>
<td>5</td>
<td>25</td>
<td>4</td>
<td>Sampson</td>
<td>05-23-05</td>
</tr>
</tbody>
</table>

CURRENT PROGRAM INDEX 97

Thus, we have arrived at a program risk index or metric. If we maintain a chart of this metric over time we see that it characteristically will rise early in a program as risks are identified but there is a delay in mitigating them causing a rising metric as shown in Figure 38. As a program progresses, this value will rise to a peak at some point in the program and subsequently start a long decline. Risks continue to be identified so we see the accumulated total number of risks continue to increase but the program is being successful in mitigating risks and later risks are commonly lower in index than those identified earlier in the program.

Figure 38  Program Risk Tracking Chart
3.5 Verification Requirements

The author respects the six-section specification structure of the DoD specification standard where Section 3 contains the product requirements and Section 4 the verification requirements. For every product requirement in Section 3 there should be one or more verification process requirements in Section 4. Whoever writes the Section 3 requirement should also write the verification requirement and with little time between the two actions. The rationale here is that the author of an unverifiable requirement will have some much difficulty writing a verification requirement and this difficulty may stimulate them to look for better ways of writing the requirement leading to a verifiable requirement.

We commonly respect four methods of verification: test, analysis, demonstration, and examination. A verification traceability matrix should be included in every specification that correlates every requirement in Section 3 with a method of verification and one or more verification requirements in Section 4. Each one of the rows in the verification traceability matrix forms a verification string. All of the verification traceability matrices are pooled into a single program-wide verification compliance matrix shown in Figure 39 that lists every verification string.

![Figure 39 Verification Traceability](image-url)
A verification engineer or team must now assign verification task numbers to all of the strings in the compliance matrix. Each task is identified in a task matrix and coordinated with the name of a principal engineer who must plan the task, make arrangements for the needed resources in a timely way relative to the task, accomplish the task on schedule, and produce a verification report. The reports from all of the tasks for a given item may be subjected to a configuration audit by the customer to ensure that the contractor did meet all of the requirements in the specification.

3.6 Specification Review and Approval

No matter the path the specification has taken through the requirements analysis process relative to modeling, the program should pass it through a review and approval process before it becomes a part of the formal configuration baseline definition. The review and approval process, shown in Figure 40, offers a formal or informal peer review way of comparing the content of a specification with a set of standards that all specifications should meet. Following approval, the specification must be formally released, published, and made available to program personnel either in paper or on-line form. The released specification must thereafter be formally protected through configuration management. Any changes must pass back through this same process to gain approval.

The formal review process should include a conscious evaluation of specification template faithfulness and overall quality measured in accordance with a specification checklist. Next, the specification should be checked for adherence with good traceability standards. The program may choose to fully implement traceability standards shown in the figure or some subset thereof. The final string of checks shown in Figure 40 assesses the specification for residual risk, completeness and excess content. A decision is reached by the reviewers followed by the review chairman calling either for corrective action or approval of the specification, if needed.

Specifications prepared on small or advanced programs may not have sufficient budget to support a formal review process. In this case, while not as supportive of a low risk approach, a
specification can be reviewed by experienced people in a less organized fashion, called a peer review. The team is assembled and asked to review the document either together or on-line at their desk followed by a group meeting to discuss the content.

The master copy of each specification must be retained and protected by an assigned authority in order to protect the integrity of the document. Once approved and released, this master must be accurately identified and protected against change. In one organization the author recalls, the master was changed during work on an engineering change proposal (ECP) but the ECP was subsequently canceled. The organization no longer had a master for the specification in effect because it had become corrupted by the change work that did not materialize. It helps to consider each specification build or change a separate campaign that results in the release of a document that will exist forever. If subsequently that document is changed, the change is built anew on the preserved baseline past.

Specifications must be readily available to personnel working a program. As they are released they must be distributed to those who need them. As they are revised the same is true of the revisions. Years ago specifications were crafted with typewriters and type setting. These were published in paper form and distributed using shoe leather and mail systems. If most of your specifications are in paper media, you may have no near term option but to place them in a paper document library from which program personnel may check them out physically. But, even if this is the current case, you should be making plans to move to an on-line specification library for cost, efficiency, and document configuration control purposes.

In a paper media, after a specification is formally released, the master must go to reproduction where sufficient copies are made to cover the needs for distribution and the library. The master should be returned to what the author has accurately heard referred to as the vault, a physically secure facility (not in the classified data sense unless this is also a valid concern) where all of the engineering masters are retained. The copies must then be physically distributed. If the specification in question is also a customer-approved specification, another loop will be required to gain their approval prior to distribution. A networked library will avoid a great deal of this busy work.

Adios paper and good riddance. Even if you are currently using a paper media for distribution you probably already have the resources in place to convert to networked computer media. It requires specifications captured in computer file media, a network with adequate storage capacity and speed, and easy access from work stations on the part of the people. These features are present in most everyone’s shop today or are not beyond the pale to achieve. It is unimaginable that anyone would use a typewriter today to prepare a specification so they will always be created in some form of word processor or computer database. The results of this work can be passed on to the document release function on a disk or via the network connection and thereafter transferred to an on-line library from which anyone may open it but not change it.
AN EFFECTIVE SPECIFICATION DEVELOPMENT ALGORITHM TUTORIAL

 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, AGM-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
An Effective Specification Development Algorithm Tutorial Outline

<table>
<thead>
<tr>
<th></th>
<th>Specification Preparation Readiness</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td></td>
</tr>
<tr>
<td>2.2</td>
<td></td>
</tr>
<tr>
<td>2.3</td>
<td>Traditional Structured Analysis</td>
</tr>
<tr>
<td>2.4</td>
<td>Traditional Structured Analysis</td>
</tr>
<tr>
<td>2.5</td>
<td>Unified Modeling Language</td>
</tr>
<tr>
<td>2.6</td>
<td>Unified Modeling Language</td>
</tr>
<tr>
<td>2.7</td>
<td>Requirements Management</td>
</tr>
<tr>
<td>2.8</td>
<td>Requirements Management</td>
</tr>
</tbody>
</table>

Success Is Possible

• The Goal
  – Good specifications
  – On time
  – Affordable

• The Plan
  – A sound beginning - be prepared
  – A clear path to a successful state - always clear what must be done
  – An effective closing - a specification review and approval process
What is a Specification?

A specification contains all of the requirements for a given item.

The Word Requirement, From The Dictionary

- Something wanted or necessary.
- Something essential to the existence or occurrence of something else.
- A necessary characteristic or attribute of something (or item).
In Writing a Specification, What Is the Target?

How to Hit the Target of Minimized Completeness

- Every performance requirement traceable to a model from which it was derived
- Every external interface for the item identified and defined in interface requirements in the specification (unless ICD applied)
- Every specialty engineering discipline that has been mapped to the item is included in the specification
- Every environmental influence defined in the appropriate model (system, end item, component) mapped to appropriate specification content.
- Every requirement in the specification traceable to a parent item specification requirement (ideally applies to the system specification relative to user requirements as well).
- Requirements are quantified as appropriate to the statement.
- Requirements are validated (risks understood and mitigated).
Product Requirements Types

- Hardware
  - Performance
  - Constraints
    - Interface
    - Specialty Engineering
    - Environmental
- One view of software requirements
  - Functional
  - Non-Functional
  - Quality
- My view of software types
  - Same as systems and hardware

Requirements Types

All of these requirements must be identified before product and process detailed design work is started and they must be mutually consistent.
Attributes of a Good Requirement

- Achievable (validated)
- Quantified
- Achievable (validated)
- Verifiable/testable
- Unambiguous
- Complete (covers all cases)
- Performance specification
  - Design independent
  - What, not how
- Detail specification
  - Design dependent

Some Good Examples

Frequency coverage. Item frequency coverage shall be 225.0 to 399.9 Megahertz inclusive in tenth Megahertz steps.

Weight. Item weight shall be less than or equal to 240 pounds.

Range. Maximum achievable range shall be greater than or equal to 5,000 nautical miles while recognizing a fuel loading safety margin of 10% or more.

Range. Maximum range shall be greater than or equal to 2,500 nautical miles.

Reliability. Item MTBF shall be greater than or equal to 10,000 hours.
Some Bad Examples

- The screens used in the system shall be designed in a user friendly manner.
- Item weight shall not be greater than 153 pounds.
- Aircraft shall identify their position within 1000 feet of actual along and across track position using Loran C.
- Brakes shall function smoothly and stop the train in a safe distance.
- There shall be no hailstorms in the path of the aircraft.
- On most days, transmitter power output should be 100 watts.
- Go fast.
- Item shall work well and last a long time.
- Any favorites from your past?

Specifications Are Full of Sentences

- THESE SENTENCES SHOULD BE WRITTEN IN THE SIMPLEST POSSIBLE WAY
- THE SUBJECT IS THE ITEM CHARACTERISTIC ABOUT WHICH THE REQUIREMENT IS WRITTEN
- VERB SHALL CLEARLY CALLS FOR COMPLIANCE

<table>
<thead>
<tr>
<th>SUBJECT</th>
<th>VERB</th>
<th>OBJECT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Item memory margin</td>
<td>shall be</td>
<td>greater than 100%</td>
</tr>
</tbody>
</table>
The Verb

- **SHALL**  Mandatory
- **WILL**  Contractor intends to perform
- **SHOULD**  Recommended, Desirable

The Subject

- Writing requirements is easy
- The difficult job is knowing what to write them **about** - the subject of the sentence
- That is why we model the problem space
Good Examples In Primitive Form

weight \leq 240 \text{ pounds}

range \geq 5,000 \text{ nautical miles}

MTBF \geq 10,000 \text{ hours}

Requirements Analysis Strategies
A Foolproof Search For Subjects

General Program Task-Resource Relationships
Work Product Development Suite
Specification Case

Document Progressions

TAILORED MIL-STD-961E

SYSTEM SPECIFICATION TEMPLATE

SYSTEM SPECIFICATION DID USING TSA

EXHIBIT B-1

SYSTEM SPECIFICATION

TAILORED MIL-STD-961E

HARDWARE ITEM PERFORMANCE SPECIFICATION TEMPLATE

HARDWARE ITEM PERFORMANCE SPECIFICATION DID USING TSA

EXHIBIT B-2

HARDWARE ITEM PERFORMANCE SPECIFICATION

JOG System Engineering, Inc.
Preparatory Steps

A Single Model Will Not Work

System Elements

Software Content

Hardware Content

A2-1-13
Hardware and Systems Analysis Models

- Traditional structured analysis
  - Functional analysis
    - Functional flow diagramming
    - Enhanced functional flow diagramming (CORE)
    - Behavioral diagramming (RDD/IPO)
    - IDEF 0 (SADT)
    - Process flow analysis
    - Hierarchical functional analysis

- Constraints analysis
  - State diagramming
  - SysML

Computer Software Structured Analysis Models

- Process-oriented analysis
  - Flow charting
  - Modern Structured Analysis (Yourdon-Demarco)
  - Modern Structured Analysis (Hatley-Pirbhai)

- Data-oriented analysis
  - Table normalizing
  - IDEF-1X

- Object-oriented analysis
  - Early models
    - Unified Modeling Language (UML)

- DoD architecture framework (DoDAF)
Structured View of a Problem Space

Structured Analysis Methods Comparison

<table>
<thead>
<tr>
<th>MULTI-FACETED APPROACHES</th>
<th>PRODUCT ENTITY FACET</th>
<th>FUNCTIONAL FACET</th>
<th>BEHAVIOR FACET</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRADITIONAL STRUCTURED ANALYSIS</td>
<td>PRODUCT ENTITY BLOCK DIAGRAM</td>
<td>FUNCTIONAL FLOW DIAGRAM</td>
<td>SCHEMATIC BLOCK DIAGRAM</td>
</tr>
<tr>
<td>MODERN STRUCTURED ANALYSIS</td>
<td>HIERARCHICAL DIAGRAM</td>
<td>DATA FLOW DIAGRAM</td>
<td>P SPEC, STATE DIAGRAM</td>
</tr>
<tr>
<td>EARLY OBJECT-ORIENTED ANALYSIS</td>
<td>CLASS AND OBJECT DIAGRAM</td>
<td>DATA FLOW DIAGRAM</td>
<td>STATE DIAGRAM</td>
</tr>
<tr>
<td>UML</td>
<td>CLASS/OBJECT, COMPONENT, &amp; DEPLOYMENT DIAGRAMS</td>
<td>USE CASES AND ACTIVITY DIAGRAMS</td>
<td>STATE, SEQUENCE, AND COMMUNICATION DIAGRAMS</td>
</tr>
</tbody>
</table>

○ UNPRECEDENTED ANALYTICAL ENTRY FACET
Model Suggestions for Today

<table>
<thead>
<tr>
<th>SPECIFICATION TYPE</th>
<th>MODEL SUGGESTED</th>
</tr>
</thead>
<tbody>
<tr>
<td>System Specification</td>
<td>Traditional Structured Analysis</td>
</tr>
<tr>
<td>Hardware Performance Specification</td>
<td>Traditional Structured Analysis</td>
</tr>
<tr>
<td>Software Performance Specification, General</td>
<td>Unified Modeling Language (UML)</td>
</tr>
<tr>
<td>Software Performance Specification, Database</td>
<td>IDEF-1X</td>
</tr>
<tr>
<td>Software Performance Specification, DoD IS</td>
<td>DoDAF</td>
</tr>
</tbody>
</table>

But, be prepared to move to the use of SysML coupled with UML and the eventual merge of the two into a more fully integrated common modeling method.

Preparatory Steps
Enabling Documentation

EDG defines the enterprise common process to be applied on all programs, the functional departments, and their process responsibilities which include preparing a department manual covering all common process work allocated to the department.

The system engineering department manual that defines the way system engineering will be applied on all programs including requirements analysis and specification management.

All functional departments should define templates and data item descriptions for all work products that are prepared as documents. One example of these artifacts is a set of specification data item descriptions that are coordinated with a particular template and a method of acquiring the content through modeling.

Preparatory Steps
### MIL-STD-961E Template

1. **Scope**  
2. **Applicable Documents**  
3. **Requirements**  
4. **Functional and Performance Rqmts.**  
5. **Missions**  
6. **Threat**  
7. **Required States and Modes**  
8. **Entity Capability Requirements**  
9. **Reliability**  
10. **Maintainability**  
11. **Deployability**  
12. **Availability**  
13. **Environmental Conditions**  
14. **Transportability**  
15. **Materials and Processes**  
16. **Electromagnetic Radiation**  
17. **Nameplates and Product Markings**  
18. **Producibility**  
19. **Interchangeability**  
20. **Safety**  
21. **Human Factors Engineering**  
22. **Security and Privacy**  

---

### Recommended Template With Map for Traditional Structured Analysis

<table>
<thead>
<tr>
<th>PARA</th>
<th>TITLE</th>
<th>MODEL</th>
<th>DEPT</th>
<th>APP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>SCOPE</td>
<td>DID</td>
<td>2000</td>
<td></td>
</tr>
<tr>
<td>2.</td>
<td>APPLICABLE DOCUMENTS</td>
<td>DID</td>
<td>2000</td>
<td></td>
</tr>
<tr>
<td>3.</td>
<td>REQUIREMENTS</td>
<td>DID</td>
<td>2000</td>
<td>A</td>
</tr>
<tr>
<td>3.1</td>
<td>Missions</td>
<td>DID</td>
<td>2000</td>
<td></td>
</tr>
<tr>
<td>3.1.1</td>
<td>Threat</td>
<td>Mission Analysis</td>
<td>2000</td>
<td></td>
</tr>
<tr>
<td>3.1.2</td>
<td>Required states and modes</td>
<td>DID</td>
<td>2000</td>
<td></td>
</tr>
<tr>
<td>3.1.3</td>
<td>Functional analysis</td>
<td>DID</td>
<td>2000</td>
<td>A</td>
</tr>
<tr>
<td>3.1.3.1</td>
<td>Specialty engineering requirements</td>
<td>DID</td>
<td>2000</td>
<td>A</td>
</tr>
<tr>
<td>3.1.3.2</td>
<td>Environmental model</td>
<td>DID</td>
<td>2000</td>
<td>B</td>
</tr>
<tr>
<td>3.2</td>
<td>Entity capability requirements</td>
<td>Functional Analysis</td>
<td>2000</td>
<td>A</td>
</tr>
<tr>
<td>3.2.m</td>
<td>Capability m</td>
<td>Functional Analysis</td>
<td>2000</td>
<td>A</td>
</tr>
<tr>
<td>3.2.m.n</td>
<td>Capability m, requirement n</td>
<td>Functional Analysis</td>
<td>2000</td>
<td>A</td>
</tr>
<tr>
<td>3.3</td>
<td>Interface requirements</td>
<td>N-Square Diagram</td>
<td>2000</td>
<td>D</td>
</tr>
<tr>
<td>3.3.1</td>
<td>External interface requirements</td>
<td>N-Square Diagram</td>
<td>2000</td>
<td>D</td>
</tr>
<tr>
<td>3.3.1.m</td>
<td>External interface m</td>
<td>N-Square Diagram</td>
<td>2000</td>
<td>D</td>
</tr>
<tr>
<td>3.3.2</td>
<td>Internal interface requirements</td>
<td>N-Square Diagram</td>
<td>2000</td>
<td>D</td>
</tr>
<tr>
<td>3.3.2.m</td>
<td>Internal interface m</td>
<td>N-Square Diagram</td>
<td>2000</td>
<td>D</td>
</tr>
<tr>
<td>3.3.2.m.n</td>
<td>Internal Interface m, requirement n</td>
<td>N-Square Diagram</td>
<td>2000</td>
<td>D</td>
</tr>
</tbody>
</table>

---

*NDIA*

**Recommended Template With Map for Traditional Structured Analysis**

<table>
<thead>
<tr>
<th>PARA</th>
<th>TITLE</th>
<th>MODEL</th>
<th>DEPT</th>
<th>APP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>SCOPE</td>
<td>DID</td>
<td>2000</td>
<td></td>
</tr>
<tr>
<td>2.</td>
<td>APPLICABLE DOCUMENTS</td>
<td>DID</td>
<td>2000</td>
<td></td>
</tr>
<tr>
<td>3.</td>
<td>REQUIREMENTS</td>
<td>DID</td>
<td>2000</td>
<td>A</td>
</tr>
<tr>
<td>3.1</td>
<td>Missions</td>
<td>DID</td>
<td>2000</td>
<td></td>
</tr>
<tr>
<td>3.1.1</td>
<td>Threat</td>
<td>Mission Analysis</td>
<td>2000</td>
<td></td>
</tr>
<tr>
<td>3.1.2</td>
<td>Required states and modes</td>
<td>DID</td>
<td>2000</td>
<td></td>
</tr>
<tr>
<td>3.1.3</td>
<td>Functional analysis</td>
<td>DID</td>
<td>2000</td>
<td>A</td>
</tr>
<tr>
<td>3.1.3.1</td>
<td>Specialty engineering requirements</td>
<td>DID</td>
<td>2000</td>
<td>A</td>
</tr>
<tr>
<td>3.1.3.2</td>
<td>Environmental model</td>
<td>DID</td>
<td>2000</td>
<td>B</td>
</tr>
<tr>
<td>3.2</td>
<td>Entity capability requirements</td>
<td>Functional Analysis</td>
<td>2000</td>
<td>A</td>
</tr>
<tr>
<td>3.2.m</td>
<td>Capability m</td>
<td>Functional Analysis</td>
<td>2000</td>
<td>A</td>
</tr>
<tr>
<td>3.2.m.n</td>
<td>Capability m, requirement n</td>
<td>Functional Analysis</td>
<td>2000</td>
<td>A</td>
</tr>
<tr>
<td>3.3</td>
<td>Interface requirements</td>
<td>N-Square Diagram</td>
<td>2000</td>
<td>D</td>
</tr>
<tr>
<td>3.3.1</td>
<td>External interface requirements</td>
<td>N-Square Diagram</td>
<td>2000</td>
<td>D</td>
</tr>
<tr>
<td>3.3.1.m</td>
<td>External interface m</td>
<td>N-Square Diagram</td>
<td>2000</td>
<td>D</td>
</tr>
<tr>
<td>3.3.2</td>
<td>Internal interface requirements</td>
<td>N-Square Diagram</td>
<td>2000</td>
<td>D</td>
</tr>
<tr>
<td>3.3.2.m</td>
<td>Internal interface m</td>
<td>N-Square Diagram</td>
<td>2000</td>
<td>D</td>
</tr>
<tr>
<td>3.3.2.m.n</td>
<td>Internal Interface m, requirement n</td>
<td>N-Square Diagram</td>
<td>2000</td>
<td>D</td>
</tr>
</tbody>
</table>

---

*NDIA*

**Recommended Template With Map for Traditional Structured Analysis**

<table>
<thead>
<tr>
<th>PARA</th>
<th>TITLE</th>
<th>MODEL</th>
<th>DEPT</th>
<th>APP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>SCOPE</td>
<td>DID</td>
<td>2000</td>
<td></td>
</tr>
<tr>
<td>2.</td>
<td>APPLICABLE DOCUMENTS</td>
<td>DID</td>
<td>2000</td>
<td></td>
</tr>
<tr>
<td>3.</td>
<td>REQUIREMENTS</td>
<td>DID</td>
<td>2000</td>
<td>A</td>
</tr>
<tr>
<td>3.1</td>
<td>Missions</td>
<td>DID</td>
<td>2000</td>
<td></td>
</tr>
<tr>
<td>3.1.1</td>
<td>Threat</td>
<td>Mission Analysis</td>
<td>2000</td>
<td></td>
</tr>
<tr>
<td>3.1.2</td>
<td>Required states and modes</td>
<td>DID</td>
<td>2000</td>
<td></td>
</tr>
<tr>
<td>3.1.3</td>
<td>Functional analysis</td>
<td>DID</td>
<td>2000</td>
<td>A</td>
</tr>
<tr>
<td>3.1.3.1</td>
<td>Specialty engineering requirements</td>
<td>DID</td>
<td>2000</td>
<td>A</td>
</tr>
<tr>
<td>3.1.3.2</td>
<td>Environmental model</td>
<td>DID</td>
<td>2000</td>
<td>B</td>
</tr>
<tr>
<td>3.2</td>
<td>Entity capability requirements</td>
<td>Functional Analysis</td>
<td>2000</td>
<td>A</td>
</tr>
<tr>
<td>3.2.m</td>
<td>Capability m</td>
<td>Functional Analysis</td>
<td>2000</td>
<td>A</td>
</tr>
<tr>
<td>3.2.m.n</td>
<td>Capability m, requirement n</td>
<td>Functional Analysis</td>
<td>2000</td>
<td>A</td>
</tr>
<tr>
<td>3.3</td>
<td>Interface requirements</td>
<td>N-Square Diagram</td>
<td>2000</td>
<td>D</td>
</tr>
<tr>
<td>3.3.1</td>
<td>External interface requirements</td>
<td>N-Square Diagram</td>
<td>2000</td>
<td>D</td>
</tr>
<tr>
<td>3.3.1.m</td>
<td>External interface m</td>
<td>N-Square Diagram</td>
<td>2000</td>
<td>D</td>
</tr>
<tr>
<td>3.3.2</td>
<td>Internal interface requirements</td>
<td>N-Square Diagram</td>
<td>2000</td>
<td>D</td>
</tr>
<tr>
<td>3.3.2.m</td>
<td>Internal interface m</td>
<td>N-Square Diagram</td>
<td>2000</td>
<td>D</td>
</tr>
<tr>
<td>3.3.2.m.n</td>
<td>Internal Interface m, requirement n</td>
<td>N-Square Diagram</td>
<td>2000</td>
<td>D</td>
</tr>
</tbody>
</table>

---

*NDIA*
Recommended Template With Map for Traditional Structured Analysis

<table>
<thead>
<tr>
<th>PARA</th>
<th>TITLE</th>
<th>MODEL</th>
<th>DEPT</th>
<th>APP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>SCOPE</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>2.</td>
<td>APPLICABLE DOCUMENTS</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.</td>
<td>REQUIREMENTS</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1</td>
<td>Functional and performance requirements</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.1</td>
<td>Missions</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.2</td>
<td>Threat</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.3</td>
<td>Required states and modes</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.3.1</td>
<td>Functional analysis</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.3.2</td>
<td>Subordinate entities</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.3.3</td>
<td>Interface relationships</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.3.4</td>
<td>Specialty engineering requirements</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.3.5</td>
<td>Environmental model</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.2</td>
<td>Entity capability requirements</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.2.m</td>
<td>Capability m</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.2.m.n</td>
<td>Capability m, requirement n</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.3</td>
<td>Interface requirements</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.3.1</td>
<td>External interface requirements</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.3.1.m</td>
<td>External interface m</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.3.1.m.n</td>
<td>External interface m, requirement n</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.3.2</td>
<td>Internal interface requirements</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.3.2.m</td>
<td>Internal interface m</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.3.2.m.n</td>
<td>Internal interface m, requirement n</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
</tbody>
</table>
Prepare and Maintain DIDs

- The DID follows the template format
- The recommended DID is focused on a particular modeling approach
- The DID tells how to create a specification using that template with a particular modeling approach

Here is a sample DID for a system specification using TSA.

Preparatory Steps
### Generic Specialty Engineering Scoping Matrix

<table>
<thead>
<tr>
<th>SPECIALTY DISCIPLINE</th>
<th>DEPT</th>
<th>PARA</th>
<th>A</th>
<th>A1</th>
<th>A2</th>
<th>A3</th>
<th>A4</th>
</tr>
</thead>
<tbody>
<tr>
<td>H1 Reliability</td>
<td>D264</td>
<td>3.4.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H2 Maintainability</td>
<td>D264</td>
<td>3.4.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H3 Availability</td>
<td>D264</td>
<td>3.4.3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H4 Deployability and transportability</td>
<td>D264</td>
<td>3.4.4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H5 Logistics</td>
<td>D231</td>
<td>3.4.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H6 Maintenance</td>
<td>D231</td>
<td>3.4.5.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H7 Interchangeability</td>
<td>D213</td>
<td>3.4.5.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H8 Supply</td>
<td>D231</td>
<td>3.4.5.3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H9 Facilities and facility equipment</td>
<td>D231</td>
<td>3.4.5.4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HA Personnel</td>
<td>D231</td>
<td>3.4.5.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HB Training</td>
<td>D213</td>
<td>3.4.5.6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HC Safety</td>
<td>D213</td>
<td>3.4.6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HD Human factors engineering</td>
<td>D210</td>
<td>3.4.7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HE Security and privacy</td>
<td></td>
<td>3.4.8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HF Electromagnetic compatibility</td>
<td>D213</td>
<td>3.4.9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HG Lightning protection</td>
<td>D213</td>
<td>3.4.10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HH Productibility</td>
<td>D213</td>
<td>3.4.11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HI Affordability</td>
<td>D213</td>
<td>3.4.12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HJ Computer resource requirements</td>
<td>3.4.13</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HK Quality Engineering</td>
<td>D5100</td>
<td>3.4.14.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HL Parts, materials, and processes</td>
<td>D2167</td>
<td>3.4.14.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Environmental Requirements

- **System**
  - Identify spaces within which the system will have to function
  - Select standards covering those spaces
  - For each standard, select parameters that apply
  - Tailor the range of selected parameters

- **End item**
  - Build three-dimensional model of end items, physical processes, and process environments
  - Extract item environments

- **Component**
  - Zone end item into spaces of common environmental characteristics
  - Map components to zones
  - Components inherit zone environmental requirements
### Specialty Engineering Scoping Matrix

#### Applied to Program

**Specialty Engineering Scoping Matrix**

<table>
<thead>
<tr>
<th>Constraint</th>
<th>A11</th>
<th>A12</th>
<th>A13</th>
<th>A14</th>
<th>A15</th>
</tr>
</thead>
<tbody>
<tr>
<td>H21</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H22</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H23</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H31</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H32</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H41</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H42</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H51</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H52</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H53</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H61</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H71</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Specialty Engineering Requirement**

**Requirements Analysis Sheet**

*(In a computer database)*

#### Configuration Control the Models

**System Definition Document**

- **APPENDIX A**, **SYSTEM FUNCTIONALITY**
- **APPENDIX B**, **SYSTEM ENVIRONMENT**
- **APPENDIX C**, **PRODUCT ENTITY**
- **APPENDIX D**, **INTERFACE**
- **APPENDIX E**, **SPECIALTY ENGINEERING**
- **APPENDIX F**, **PROCESS**
- **APPENDIX G**, **RAS**
- **APPENDIX H**, **MANPOWER**
On To Program Work
THE BIG BANG THEORY OF SYSTEM DEVELOPMENT
THE TRADITIONAL APPROACH

EVERYTHING FLOWS FROM ONE IDEA,
THE CUSTOMER NEED
IT IS THE ULTIMATE REQUIREMENT,
THE ULTIMATE FUNCTION

BA-BA-BA-BANG
Two Top-Level Views of a System

The Beginning Of Functional Decomposition
Traditional Structured Analysis Model

Overview

1. Understand User Requirements
2. Functional Flow Diagram
3. Performance Requirements Analysis
4. Requirement Allocation
5. Environmental Requirements Analysis
6. Product Entity Structure
7. Requirements Analysis Sheet
8. N-Square Diagram
9. Specifications Cycle to Lower Tiers
10. Special Engineering Requirements Analysis
11. Technical Engineering Requirements Analysis

The Ultimate Function and Its First Expansion

First Expansion is a Life-Cycle Flow Diagram

Alternative functional analysis techniques
- Enhanced functional flow block diagramming (CORE)
- Behavioral diagramming (RDD)
- IDEF-0
Example of a Life Cycle Model

Use System Expansion Example
Space Transport System
Continued Function Decomposition

An orderly exposure of needed functionality moving from the known to the unknown, from simple to the complex, from the top to the bottom.

Performance Requirements Analysis and the RAS

<table>
<thead>
<tr>
<th>FUNCTION</th>
<th>PRODUCT ENTITY</th>
<th>REQUIREMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>MID</td>
<td>NAME</td>
<td>PID</td>
</tr>
<tr>
<td>F123</td>
<td>Provide Thrust</td>
<td>A12</td>
</tr>
</tbody>
</table>

Exposing what the system must do and how well it must do it encouraging identification of all essential characteristics and avoidance of unnecessary characteristics.
The Function-Product Entity Plane

Functional Analysis Alternatives

- **IDEF 0**
  - A variation on SADT
- **Behavioral Diagramming**
  - From Ascent Logic's RDD
  - Based on IPO
- **Enhanced Functional Flow Block Diagramming**
  - Employed in Vitech's CORE
- **Hierarchical Functional Block Diagramming**
Requirements Capture Using the RAS-Complete Format

Here Is What We Want

This Is How To Get It

Product Entity Structure

Use a common structure that includes hardware and software.
Systems Consist of Things and Relationships

Organizing For Interface Development

- Decompose needed functionality and allocate to product entities
- Map product entities to responsible development organizations
  - Create cross-functional integrated product and process teams (IPPT)
  - Assign principal engineers for lowest tier responsibilities on teams (everything has someone responsible)
- Establish clear rules for interface development responsibility
  - Identify needed interfaces as a function of how functionality was allocated to entities
  - Analyze product entity pair relationships using n-square diagrams
  - Partition interface into subsets as a function of product entity principal views
  - Assign interface responsibility to product entity principal engineers as a function of a receiving terminal rule (if you need an interface you must come forward)
  - System engineering manage the aggregate external and inter-team interface sets applying a lowest common team integration concept
- Minimize external (cross-organizational) interface at all levels, iterating product entity structure and/or development organization responsibilities to do so, if necessary, then apply system engineering integration resources to that which remains
Two Interface Definition Models

SCHEMATIC BLOCK DIAGRAMMING

- Lines define interfaces
- Blocks are objects only from the product entity structure

N-SQUARE DIAGRAMMING

- Marked intersections define interfaces
- Diagonal blocks are objects only from product entity block diagram
- Apparent ambiguity reflects directionality

Interface Requirements Derivation

Geometrical View
Development Often Fails at the Cross-organizational Interfaces

Interface Integration Focus
The Fundamental Problems in Interface Work

There is a one-to-one correspondence between teams and components. There is a one- to-two correspondence between teams and interfaces.

We tend to focus inwardly

COMPONENT X

COMPONENT Y

COMPONENT X

COMPONENT Y
The Fundamental Problems in Interface Work

We are dependent on the worst interface on planet Earth in the development of interfaces.

Benefits Of Product Team Organization
Product Entity and Interface Responsibility

Lowest Common Team Concept
Specialty Engineering Identification of Constraints

SPECIALTY ENGINEERING CONSTRAINTS MATRIX

SPECIALTY ENGINEERING REQUIREMENTS FLOW INTO THE INDICATED SPECIFICATIONS VIA THE RAS IMPLEMENTED IN A DATABASE

Specialty Engineering Plane Added
Environment Subsets

Environmental Requirements

- **System**
  - Identify spaces within which the system will have to function
  - Select standards covering those spaces
  - For each standard, select parameters that apply
  - Tailor the range of selected parameters

- **End item**
  - Build three dimensional model of end items, physical processes, and process environments
  - Extract item environmental requirements

- **Component**
  - Zone end item into spaces of common environmental characteristics
  - Map components to zones
  - Components inherit zone environmental requirements
Environmental Planes Added

RAS Complete In Tabular Form
Save the Models!
AN EFFECTIVE SPECIFICATION DEVELOPMENT ALGORITHM TUTORIAL

UNIFIED MODELING LANGUAGE

Agenda

- The Dead Sea scrolls of software development
- UML modeling artifacts
- UML modeling approach
- Integration
- Requirements and modeling documentation
Software Dead Sea Scrolls

- Flow chartings
  - Functionality examined
  - Data in the back seat
- SADT and IPO
  - Two axis models
- Modern structured analysis and HP
  - Functionality and data examined
- Early OOA
  - Search for objects and their behavior
  - Anti Sullivan

A Preferred Modeling Order

Early object oriented analysis encouraged this pattern.

We will follow Sullivan’s encouragement in this tutorial - form follows function.

UML can support either direction.

Note: A classifier is a general term for a software product entity represented by a node, component, or class in UML.
The Software Development Process

- Identify a product entity that will be developed as computer software.
- Dynamically analyze the entity.
  - Use cases
  - Sequence diagram
  - Communication diagram
  - activity diagram
  - State diagram
- In the sequence, communication, and activity diagramming analysis you will have to identify lower tier 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.

Consolidated System Product Entity Structure
Suggested SRS Structure

3 REQUIREMENTS
3.1 Required states and modes
3.2 Software entity capability requirements
3.2.m Software entity capability m
3.2.m.n Software entity capability m, requirement n
3.3 Software entity interface requirements
3.3.1 Software entity external interface requirements
3.3.1.m Specific external interface m
3.3.1.m.n External interface m, requirement n
3.3.2 Software entity internal interface requirements
3.3.2.m Specific internal interface m
3.3.2.m.n Internal interface m, requirement n
3.3.3 Software entity internal data requirements
3.3.3.n Specific software entity internal data requirement n
3.4 Specialty engineering requirements
3.5 Software entity environmental requirements
3.6 Precedence and criticality requirements

The Diagrams of UML

- For modeling dynamic aspects of the system
  - Use case diagram
  - Sequence diagram
  - Timing diagram
  - Communication diagram (renamed in 2)
  - State diagram
  - Activity 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)

(2) = added in UML 2.0

Covered in this tutorial
The Dynamic Models

Sequence Diagram UX11321

Activity Diagram UX11323

Interaction Diagrams

Communication Diagram UX11322

State Diagram UX11324

Sequence Diagram UX-11321

Emphasizes the time ordering of messages

Actor

Classifier AX

a:Classifier AX1

b:Classifier AX2

messageOne()

messageTwo()

messageFour()

messageThree()

Argument List

Lifeline active

Time

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

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

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
A Simple Example

Communication Diagram UX11322
Emphasizes structural relationships

Semantically identical to the sequence diagram.
Activity Diagram UX11323

State Diagram UX11324
The Static Entities in UML

- **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 node or collection of nodes.

- **Node**
  - Appears on a deployment diagram that exists at run time and 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 classes.

- **Class**
  - A description of a set of objects that share the same attributes, operations, relationships, and semantics.

- **Object**
  - An instance of a class.

---

UML Structural Artifacts in a Product
Entity Structure

Top-Down Development Encouraged

Dynamic Analysis

Lower Tier Classifiers Identified
Deployment and Component Diagrams

Classes and Objects

A class is a description of a set of objects that share the same attributes, operations, relationships, and semantics. An object is an instance of a class. Graphically a class is rendered as a rectangle.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>The name is a noun or noun phrase</td>
</tr>
<tr>
<td>attributeOne</td>
<td>An attribute is a named property of a class that describes a range of values that instances of the property may hold. An attribute represents some property of the thing you are modeling that is shared by all objects of that class.</td>
</tr>
<tr>
<td>attributeTwo</td>
<td>An operation is the implementation of a service that can be requested from any object of a class to effect behavior. An operation is an abstraction of something you can do to an object that is shared by all objects of that class</td>
</tr>
<tr>
<td>attributeThree</td>
<td></td>
</tr>
<tr>
<td>attributeFour</td>
<td></td>
</tr>
<tr>
<td>operationOne()</td>
<td></td>
</tr>
<tr>
<td>operationTwo()</td>
<td></td>
</tr>
<tr>
<td>operationThree()</td>
<td></td>
</tr>
</tbody>
</table>
Class Responsibilities

A responsibility is a contract or an obligation of a class. You may find it useful to begin the analysis of classes this way translating these into attributes and operations that best fulfill the class’s responsibility as the model is refined. The responsibility is noted in an added compartment in which descriptive free form-text is entered.

Structural Relationships

These Have Nothing To Do With Messages

NameOne is the base class. NameTwo and NameThree are leaf classes in a generalization.

NameOne depending on NameFour for information and services.

An association is a structural relationship
Structural Relationships
Association Adornments

- Association Name
  - NameOne
  - NameTwo

- Association Role
  - The face that the class at the far end of an association presents to the class at near end of the association. Role names called end names.

- Association Multiplicity
  - Tells how many objects may be connected across an association instance. Given by a range of numbers.

- Association Aggregation
  - Expresses a whole-part relationship between to associated classes.

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 use case.
- For each scenario build a sequence diagram and in the process identify next lower tier classifiers and messages between the actors and lower tier classifiers.
- Apply communication, activity, and state diagrams as needed.
- Derive requirements from dynamic modeling artifacts.

Hierarchical Modeling Relationships

- A top-level software product entity
- Classifier AX
- Context Diagram Terminators
- Use Case
- Extended Use Case
- Scenario
- Sequence Diagram UX-ijk1
- Communication Diagram UX-ijk2
- Activity Diagram UX-ijk3
- State Diagram UX-ijk4
Context Diagram

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

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

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

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

TERMINATOR UX-1

TERMINATOR UX-2

TERMINATOR UX-3

CLASSIFIER AX

Use Case Fundamentals

• A use case is a more expressive context diagram common 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

- **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 UX-11

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.
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.

Examine Each Scenario Dynamically

• Activity, sequence, and communication diagrams require identification of lower tier entities leading to additional of entities on the consolidated product entity diagram
• State diagrams may also be useful in identifying essential characteristics appropriate for the entity being analyzed
• Requirements flow out of the dynamic analysis and into the specification for the entity being analyzed
AN EFFECTIVE SPECIFICATION DEVELOPMENT ALGORITHM TUTORIAL

REQUIREMENTS MANAGEMENT

Agenda

- Process summary
- Organizing and specification responsibility
- Requirements risk management
- Traceability
- Tools
- Review, approval, release, and distribution
- A peek into the future
Life Cycle Model

Suggested Enterprise Structure

- Manage the Enterprise
- Provide Program Resources
- Assure Product and Process Quality
- Acquire New Business

Perform Enterprise Business

Product Oriented Teams
Benefits of Product-Oriented Team Structure

The System Product Entity Structure and Teaming
Risk Defined

- The danger that injury, damage, or loss will occur
- Somebody or something likely to cause injury, damage, or loss
- The probability, amount, or type of possible loss incurred by an insurer
- The possibility of loss in an investment or speculation
- The statistical chance of danger from something, especially from the failure of an engineered system

Risk Measurement Parameters

<table>
<thead>
<tr>
<th>Probability of Occurrence</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAT TITLE</td>
<td>P(O)</td>
</tr>
<tr>
<td>---</td>
<td>----</td>
</tr>
<tr>
<td>5</td>
<td>Nearly Certain</td>
</tr>
<tr>
<td>4</td>
<td>Probable</td>
</tr>
<tr>
<td>3</td>
<td>Possible</td>
</tr>
<tr>
<td>2</td>
<td>Unlikely</td>
</tr>
<tr>
<td>1</td>
<td>Nearly Impossible</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Seriousness of Effect</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAT TITLE</td>
<td>DESCRIPTION</td>
</tr>
<tr>
<td>---</td>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td>5</td>
<td>Catastrophic</td>
</tr>
<tr>
<td>4</td>
<td>Serious</td>
</tr>
<tr>
<td>3</td>
<td>Moderate</td>
</tr>
<tr>
<td>2</td>
<td>Minor</td>
</tr>
<tr>
<td>1</td>
<td>Null</td>
</tr>
</tbody>
</table>
## Risk Metric Values

<table>
<thead>
<tr>
<th>SERIOUSNESS OF THE EFFECT</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>25</td>
<td>20</td>
<td>15</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>20</td>
<td>16</td>
<td>12</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>15</td>
<td>12</td>
<td>9</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>8</td>
<td>6</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

- High Risk
- Medium Risk
- Low Risk

## Aggregate Program Risk Index

![Graph showing the Aggregate Program Risk Index](image)

PEAK RISK POINT

**Program Risk Index**

**Number of Risks**

**Time**
Traceability Forms

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

Traceability Integration

- **Lateral traceability within TSA and UML** independently should be no problem with all requirements derived from models
- **Vertical interface traceability** is decomposition driven within TSA and UML as well as across the HW-SW gap
- Environmental requirements are significantly different between HW and SW so traceability is not a significant issue between them
- **Performance requirements across the HW-SW gap** offer a vertical traceability challenge
The System Product Entity Structure

This is the kind of relationship of interest

Level at which a subordinate software entity is identified

Software entity

Hardware entity

System

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

Alternatively, one could rely upon experienced inspection without the organizing influence of the matrix.
Today’s Tools

Tools Integration
Tomorrow’s Tools

- Front end modeling tools
  - Use case modeling
  - Function/activity modeling
  - State modeling (behavioral modeling)
  - Sequence/timeline modeling
  - Product entity and interface modeling
  - Specialty engineering database linkage
  - Environmental coverage

- Connection of modeling to management database

- Big dumb database
  - Requirements capture
  - Traceability
  - Value management
  - Specification publishing

Tools Integration
Integrated Tool Set

Configuration Management of Requirements Documentation

Portion of database corresponding to released specifications and library content under formal configuration control.
Utility Of Computer Projection

- On-line network capability
- Put the projection capability in the work area
- Apply real-time concurrent development (IPD)
- Form and reform between meeting and individual work quickly

Specification Review and Approval Process
Evaluate for Template Faithfulness

- Compare specification cover data with template (standard)
- Compare specification paragraphing structure with template
- Compare specification style with template style guide

Individual Requirement Quality

- Spot check specification requirements for requirements quality checklist compliance
- Spot check specification for requirements quantification where appropriate
Section 2 Traceability

- All documents listed in Section 2 called somewhere in the specification
- All documents tailored, if necessary, to limit coverage to the application
- All documents called in the requirements listed in Section 2
- Spot check for excessively tailored standards which could be quoted instead of being called
- Ensure documents called are current and accepted authorities for the application

Completeness and Avoidance of Unnecessary Content

- Ask principal engineer how content was derived
  - If ad hoc, there should be concern
  - If through structured analysis, spot check how a few requirements were derived (ask to see the supporting modeling data)
- Ensure all requirements traceable to parent requirements
Residual Risk Evaluation

- All TBD/TBR are closed or, if not, are being carried as program or team risks
- An approved concept exists
UML and Functional Analysis

Unified Modeling Language (UML)

<table>
<thead>
<tr>
<th>STATIC DIAGRAMS</th>
<th>DYNAMIC DIAGRAMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEPLOYMENT DIAGRAM</td>
<td>USE CASE DIAGRAM</td>
</tr>
<tr>
<td>COMPONENT DIAGRAM</td>
<td>ACTIVITY DIAGRAM</td>
</tr>
<tr>
<td>OBJECT &amp; CLASS DIAGRAMS</td>
<td></td>
</tr>
<tr>
<td>STATE CHART</td>
<td>INTERACTION DIAGRAM</td>
</tr>
<tr>
<td>ARCHITECTURE BLOCK DIAGRAM</td>
<td>SCHEMATIC BLOCK DIAGRAM</td>
</tr>
<tr>
<td>STATE DIAGRAM</td>
<td>TIMELINE DIAGRAM</td>
</tr>
<tr>
<td>PHYSICAL FACET</td>
<td>FUNCTIONAL FLOW DIAGRAM</td>
</tr>
<tr>
<td>BEHAVIORAL FACET</td>
<td></td>
</tr>
<tr>
<td>FUNCTIONAL FACET</td>
<td></td>
</tr>
</tbody>
</table>

Traditional Structured Analysis
A Subset of UML?

Modeling Changes In the Near Term

Component Diagram
Deployment Diagram
Communication Diagram
Interaction Overview Diagram
Package Diagram

UNIVERSAL MODEL OF THE FUTURE

UML

SysML

SysML DERIVED FROM UML

Requirements Diagram
Parametric Diagram
Assembly Diagram

REPLACING TSA

PUSH THESE COMPONENTS TOGETHER MORE TIGHTLY
System Modeling Evolution Timeline

RISE IN THE USE OF STRUCTURED ANALYSIS
MODEL DRIVEN DEVELOPMENT
DATABASE DRIVEN DEVELOPMENT
DOCUMENT DRIVEN DEVELOPMENT

1920 1970 1990 2010 2030

NOW

Over the Hill and Through the Woods to Utopia

Flow Charting
Modern Structured Analysis
Early OOA
HP

1950s

Traditional Structured Analysis

Traditional Structured Analysis

1950s

UML
DoDAF
SysML

2010s

Utopia

DOG System Engineering, Inc.
Review and Summary

The target is completeness and avoidance of unnecessary content.

Use models to identify essential characteristics.

Do the analysis:
- Write requirements in primitive form based on essential characteristics identified through modeling.
- Translate into full sentences and insert into paragraph structure.
GRAND SYSTEMS
DEVELOPMENT TRAINING PROGRAM
VERSION 10.1

The union of system engineering, domain engineering, functional management, and program management for the greater good of the enterprise and customer base.

VOLUME 2R
AN EFFECTIVE SPECIFICATION DEVELOPMENT ALGORITHM

Presented By
Jeffrey O. Grady

JOG SYSTEM ENGINEERING, Inc.

6015 Charae Street
San Diego, California 92122
(858) 458-0121
(858) 456-0867 Fax
jgrady@ucsd.edu or jeff@jogse.com
http://www.jogse.com

Copyright 2006

No part of this manual may be scanned or reproduced in any form without permission in writing from the author.
<table>
<thead>
<tr>
<th>PARAGRAPH</th>
<th>TITLE</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Specification Templates and DIDs</td>
<td>2</td>
</tr>
<tr>
<td>1.1</td>
<td>Enterprise Engineering Work</td>
<td>2</td>
</tr>
<tr>
<td>1.2</td>
<td>System Engineering Generic Work</td>
<td>3</td>
</tr>
<tr>
<td>1.3</td>
<td>Proposal Work That Prepares for Program Execution</td>
<td>4</td>
</tr>
<tr>
<td>1.4</td>
<td>Work Subsequent to Contract Award</td>
<td>6</td>
</tr>
<tr>
<td>1.5</td>
<td>The Preferred Templates</td>
<td>7</td>
</tr>
<tr>
<td>1.6</td>
<td>Modeling Work Product Capture Document</td>
<td>9</td>
</tr>
<tr>
<td>2</td>
<td>Structured Analysis</td>
<td>9</td>
</tr>
<tr>
<td>2.1</td>
<td>Traditional Structured Analysis</td>
<td>9</td>
</tr>
<tr>
<td>2.1.1</td>
<td>A System Defined</td>
<td>11</td>
</tr>
<tr>
<td>2.1.2</td>
<td>The System Environment</td>
<td>11</td>
</tr>
<tr>
<td>2.1.3</td>
<td>System Functionality</td>
<td>12</td>
</tr>
<tr>
<td>2.1.4</td>
<td>Performance Requirements Derivation and Allocation</td>
<td>15</td>
</tr>
<tr>
<td>2.1.5</td>
<td>Product Entity Structure</td>
<td>15</td>
</tr>
<tr>
<td>2.1.6</td>
<td>Allocation Pacing Alternatives</td>
<td>17</td>
</tr>
<tr>
<td>2.1.7</td>
<td>System Relations</td>
<td>18</td>
</tr>
<tr>
<td>2.1.8</td>
<td>Environmental Relation Algorithm</td>
<td>20</td>
</tr>
<tr>
<td>2.1.8.1</td>
<td>System Environmental Relations</td>
<td>20</td>
</tr>
<tr>
<td>2.1.8.2</td>
<td>End Item Service Use Profile</td>
<td>20</td>
</tr>
<tr>
<td>2.1.8.3</td>
<td>Component Environmental Relations</td>
<td>21</td>
</tr>
<tr>
<td>2.1.8.4</td>
<td>Environmental Impact</td>
<td>22</td>
</tr>
<tr>
<td>2.1.9</td>
<td>Specialty Engineering and RAS Complete</td>
<td>22</td>
</tr>
<tr>
<td>2.1.10</td>
<td>RAS-Complete in Table Form</td>
<td>24</td>
</tr>
<tr>
<td>2.1.11</td>
<td>Traditional Structured Analysis Summary</td>
<td>25</td>
</tr>
<tr>
<td>2.1.12</td>
<td>SDD Content and Format</td>
<td>26</td>
</tr>
<tr>
<td>2.1.12.1</td>
<td>Document Main Body</td>
<td>27</td>
</tr>
<tr>
<td>2.1.12.2</td>
<td>Appendix A, Functional Analysis</td>
<td>27</td>
</tr>
<tr>
<td>2.1.12.3</td>
<td>Appendix B, System Environment Analysis</td>
<td>27</td>
</tr>
<tr>
<td>2.1.12.4</td>
<td>Appendix C, System Architecture Analysis</td>
<td>28</td>
</tr>
<tr>
<td>2.1.12.5</td>
<td>Appendix D, System Interface Analysis</td>
<td>28</td>
</tr>
<tr>
<td>2.1.12.6</td>
<td>Appendix E, Specialty Engineering Definition Analysis</td>
<td>28</td>
</tr>
<tr>
<td>2.1.12.7</td>
<td>Appendix F, System Process Analysis</td>
<td>28</td>
</tr>
<tr>
<td>2.1.12.8</td>
<td>Appendix G, Requirements Analysis Sheet</td>
<td>28</td>
</tr>
<tr>
<td>2.1.13</td>
<td>Team Activity During Requirements Work</td>
<td>29</td>
</tr>
<tr>
<td>2.2</td>
<td>UML</td>
<td>30</td>
</tr>
<tr>
<td>2.2.1</td>
<td>Entry Analysis and Overview</td>
<td>30</td>
</tr>
<tr>
<td>2.2.2</td>
<td>The Connection Between Modeling Artifacts, Specification, Content, and Product Entities</td>
<td>32</td>
</tr>
<tr>
<td>2.2.3</td>
<td>Dynamic Modeling Artifacts Explained</td>
<td>35</td>
</tr>
<tr>
<td>2.2.3.1</td>
<td>Sequence Diagram</td>
<td>35</td>
</tr>
<tr>
<td>2.2.3.2</td>
<td>Communication Diagram</td>
<td>37</td>
</tr>
<tr>
<td>2.2.3.3</td>
<td>Activity Diagram</td>
<td>37</td>
</tr>
</tbody>
</table>
TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>PARAGRAPH</th>
<th>TITLE</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.2.3.4</td>
<td>State Diagram</td>
<td>38</td>
</tr>
<tr>
<td>2.2.4</td>
<td>Structural Analysis</td>
<td>39</td>
</tr>
<tr>
<td>2.2.4.1</td>
<td>The Class</td>
<td>40</td>
</tr>
<tr>
<td>2.2.4.2</td>
<td>Class Relationships</td>
<td>41</td>
</tr>
<tr>
<td>2.2.4.3</td>
<td>Messages</td>
<td>42</td>
</tr>
<tr>
<td>2.2.5</td>
<td>Related Analyses</td>
<td>42</td>
</tr>
<tr>
<td>2.2.5.1</td>
<td>Specialty Engineering</td>
<td>42</td>
</tr>
<tr>
<td>2.2.5.2</td>
<td>Environmental Requirements</td>
<td>42</td>
</tr>
<tr>
<td>2.2.6</td>
<td>Specification Structure</td>
<td>42</td>
</tr>
<tr>
<td>2.2.7</td>
<td>Software Requirements Close-out</td>
<td>44</td>
</tr>
<tr>
<td>2.3</td>
<td>Opening the Analysis With DoDADF</td>
<td>45</td>
</tr>
<tr>
<td>2.4</td>
<td>Integrated Modeling</td>
<td>47</td>
</tr>
<tr>
<td>3.</td>
<td>Requirements Management</td>
<td>51</td>
</tr>
<tr>
<td>3.1</td>
<td>Summary of Team Activity During Requirements Work</td>
<td>51</td>
</tr>
<tr>
<td>3.2</td>
<td>Requirements Tools Base</td>
<td>52</td>
</tr>
<tr>
<td>3.3</td>
<td>Recommended Specification Responsibility Pattern</td>
<td>53</td>
</tr>
<tr>
<td>3.4</td>
<td>Requirements Risk Management</td>
<td>54</td>
</tr>
<tr>
<td>3.4.1</td>
<td>Requirements Validation</td>
<td>54</td>
</tr>
<tr>
<td>3.4.2</td>
<td>Margins and Budgets</td>
<td>55</td>
</tr>
<tr>
<td>3.4.3</td>
<td>Risk Tracking</td>
<td>55</td>
</tr>
<tr>
<td>3.5</td>
<td>Verification Requirements</td>
<td>58</td>
</tr>
<tr>
<td>3.6</td>
<td>Specification Review and Approval</td>
<td>59</td>
</tr>
<tr>
<td>A</td>
<td>APPENDIX A, PRESENTATION MATERIALS</td>
<td>A-i</td>
</tr>
<tr>
<td>B</td>
<td>APPENDIX B, SPECIFICATION DATA ITEM DESCRIPTIONS</td>
<td>B-i</td>
</tr>
<tr>
<td></td>
<td>JOGSE System Specification Data Item Description</td>
<td>B-1-1</td>
</tr>
<tr>
<td></td>
<td>JOGSE Hardware Item Performance Specification Data Item Description</td>
<td>B-2-1</td>
</tr>
<tr>
<td></td>
<td>JOGSE Software Requirements Specification Data Item Description</td>
<td>B-3-1</td>
</tr>
</tbody>
</table>

NOTE

Exhibit B available from the lecturer by sending an email to jgady@ucsd.edu and requesting it.
# LIST OF ILLUSTRATIONS

<table>
<thead>
<tr>
<th>FIGURE</th>
<th>TITLE</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Overall Process</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>Preparatory Steps</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>Proposal Team Work</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>Program Work</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>System and Hardware Specification Template</td>
<td>8</td>
</tr>
<tr>
<td>6</td>
<td>Overview of the Traditional Structured Analysis Process</td>
<td>10</td>
</tr>
<tr>
<td>7</td>
<td>Ultimate System Diagram</td>
<td>11</td>
</tr>
<tr>
<td>8</td>
<td>System Environment</td>
<td>12</td>
</tr>
<tr>
<td>9</td>
<td>System Context Diagram</td>
<td>12</td>
</tr>
<tr>
<td>10</td>
<td>Function Sequence</td>
<td>13</td>
</tr>
<tr>
<td>11</td>
<td>Function Decomposition</td>
<td>14</td>
</tr>
<tr>
<td>12</td>
<td>System Life Cycle</td>
<td>14</td>
</tr>
<tr>
<td>13</td>
<td>Traditional Requirements Analysis Sheet</td>
<td>15</td>
</tr>
<tr>
<td>14</td>
<td>Product Entity Structure</td>
<td>16</td>
</tr>
<tr>
<td>15</td>
<td>Juxtaposition of RAS and N-square Diagrams</td>
<td>18</td>
</tr>
<tr>
<td>16</td>
<td>A Geometric View of the RAS Complete</td>
<td>23</td>
</tr>
<tr>
<td>17</td>
<td>RAS-Complete in Tabular Form</td>
<td>25</td>
</tr>
<tr>
<td>18</td>
<td>The System Relationship</td>
<td>26</td>
</tr>
<tr>
<td>19</td>
<td>SDD Structure</td>
<td>27</td>
</tr>
<tr>
<td>20</td>
<td>Context Diagram</td>
<td>30</td>
</tr>
<tr>
<td>21</td>
<td>Unified Modeling Language Overview</td>
<td>31</td>
</tr>
<tr>
<td>22</td>
<td>Hierarchical Relationship Between UML Dynamic Modeling Artifacts</td>
<td>33</td>
</tr>
<tr>
<td>23</td>
<td>Sequence Diagram Example</td>
<td>36</td>
</tr>
<tr>
<td>24</td>
<td>Communication Diagram Example</td>
<td>37</td>
</tr>
<tr>
<td>25</td>
<td>Activity Diagram Example</td>
<td>38</td>
</tr>
<tr>
<td>26</td>
<td>State Diagram Example</td>
<td>39</td>
</tr>
<tr>
<td>27</td>
<td>The UML Classifiers</td>
<td>40</td>
</tr>
<tr>
<td>28</td>
<td>Structural Relationships</td>
<td>41</td>
</tr>
<tr>
<td>29</td>
<td>Association Adornments</td>
<td>41</td>
</tr>
<tr>
<td>30</td>
<td>Software requirements specification template</td>
<td>43</td>
</tr>
<tr>
<td>31</td>
<td>Evolving Product Entity Structure</td>
<td>45</td>
</tr>
<tr>
<td>32</td>
<td>DoDAF Development Sequence</td>
<td>46</td>
</tr>
<tr>
<td>33</td>
<td>Requirements Traceability Across the Gap</td>
<td>49</td>
</tr>
<tr>
<td>34</td>
<td>Modeling Over the Years</td>
<td>50</td>
</tr>
<tr>
<td>35</td>
<td>The Approaching Merge</td>
<td>51</td>
</tr>
<tr>
<td>36</td>
<td>Tools Environment</td>
<td>53</td>
</tr>
<tr>
<td>37</td>
<td>Risk Matrix</td>
<td>56</td>
</tr>
<tr>
<td>38</td>
<td>Program Risk Tracking Chart</td>
<td>56</td>
</tr>
<tr>
<td>39</td>
<td>Verification Traceability</td>
<td>57</td>
</tr>
<tr>
<td>40</td>
<td>Specification Review and Approval</td>
<td>59</td>
</tr>
</tbody>
</table>
# LIST OF TABLES

<table>
<thead>
<tr>
<th>TABLE</th>
<th>TITLE</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Independent and Combined SDD Appendices</td>
<td>44</td>
</tr>
<tr>
<td>2</td>
<td>Risk Probability of Occurrence Criteria</td>
<td>56</td>
</tr>
<tr>
<td>3</td>
<td>Risk Effects Criteria</td>
<td>56</td>
</tr>
<tr>
<td>4</td>
<td>Program Risk List</td>
<td>57</td>
</tr>
</tbody>
</table>
An Effective Specification Development Algorithm

The first order of business on any new program is the identification of requirements that collectively define the problem to be solved. This work can be completed in a timely and affordable way producing a quality definition of the problem space in terms of the minimum collection of requirements each one of which defines an essential characteristic of the system, or item thereof, to be developed. The target we should shoot for is all of the essential characteristics identified (no missed essential characteristics) and no unessential characteristics identified. Success in this process can be encouraged by the enterprise developing a set of specification templates and a corresponding set of data item descriptions coordinated with models selected to accomplish the requirements work and an effective suite of tools within which to capture the results. Every requirement that appears in every specification should be traceable to a model artifact from which it was derived. Before embarking on a new program, the enterprise needs to have selected this preferred set of models and tools and trained its staff to employ those models effectively entering the results in the tools suite. Every specification released is printed from the tools suite and should pass through a formal review and approval process as should any subsequent changes to any specification. As the models create work products, commonly simple diagrams, they should be saved in an organized fashion in paper or computer medias and formally released so as to be available for future system phases and modification projects.

Figure 1 offers a view of the overall process within which the preparatory work and definition activity that is discussed in the first section falls. Before entering a program, the enterprise should have prepared itself for the requirements work that will have to be done. The enterprise needs models to apply for the cases where the product is going to be implemented in computer software and in hardware. Both cases are covered in this tutorial followed by a discussion of integrated modeling.

Figure 1 Overall Process
1. Specification Templates and DIDs

Many system development organizations experience some difficulty in clearly identifying appropriate requirements for inclusion in specifications they must develop and they find it difficult to accomplish the work in an affordable and timely fashion. Over a period of the last two years the author developed an algorithm for improving system development organization ability in this work using templates and specially developed data item descriptions (DIDs). It requires some work to prepare the functional organization to support programs and some work on the part of proposal teams to accomplish initial analyses and extend the templates made available from the functional system engineering department to provide program-ready data, and work by the program teams starting with contract award and running through the period of time while specifications are being developed.

The goal of this specification algorithm is to provide for affordable and timely enterprise and program definition and documentation of new product technical requirements, the management and maintenance of related data, and publication and subsequent configuration control of the resulting documents.

The work required to implement the algorithm can be described in three preparatory steps: (1) enterprise engineering work, (2) system engineering generic work, and (3) proposal team work. The first two steps are illustrated in Figure 2. The numbers in the blocks coordinate with the steps in the algorithm. Recommended functional department responsibility for accomplishing the indicated tasks is noted at the lower left corner of the task blocks.

![Figure 2 Preparatory Steps](image)

1.1 Enterprise Engineering Work

Identify and staff an enterprise integration team (EIT) that is responsible for engineering the enterprise common process and acting as the process integration and optimization agent during its development and implementation on programs. The EIT should report to the enterprise executive.
1.1.1 The enterprise, through the efforts of the EIT, must develop a common process diagram that generically identifies all program work at a level of indenture that is adequate for making clear what work must be commonly done and allocating the corresponding work responsibilities to functional departments responsible for supplying programs with the necessary resources to accomplish that work well.

1.1.2 Allocate all work on the common process diagram to functional departments forming a task allocation matrix. This matrix establishes the requirements work that functional departments must be prepared to do on programs and any training that the functional departments are funded for and capable of performing must be focused on these tasks. This matrix covers the whole enterprise capability but in this section we are focused on doing the requirements work.

1.1.3 Functional departments collect all work allocated to them and build department manuals that explain what work must be done and provide links or descriptions for how to perform this work. One of the departments will be system engineering that will have responsibility for specification development and management on programs. For each task a functional department has responsibility, that department must identify work products that will result as a function of having accomplished the work on a program. EIT must integrate and optimize the evolving functional department work descriptions and work product identifications to ensure overall efficiency and effectiveness. All work products must have at least one user. Work products must be linked to the common process diagram tasks. Specifications are an example of a task work product and the work product of interest in this algorithm.

1.2 System Engineering Generic Work

1.2.1 To the extent that work products are documents, the responsible functional department must prepare a template containing the basic structure of the document in terms of generic paragraphing structure and calls for graphical images. In preparing for implementation of specification development and management work on programs in general, the functional system engineering department will select the specification standards that will be applied respecting the common customer base of the enterprise. They will review these standards and associated data item descriptions (DID), ensure that the system engineering department manual adequately covers specification standards the enterprise has chosen to respect, and build a set of specification templates (paragraph numbering and titles only), one for each kind of specification that will commonly have to be prepared on programs.

1.2.2 For each specification template defined, the system engineering department will determine one or more preferred modeling approaches for each kind of requirement in the template. The modeling approaches encouraged are the following relative to the primary kinds of specifications that will have to be developed:

<table>
<thead>
<tr>
<th>System Specification</th>
<th>Traditional Structured Analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hardware Performance Specification</td>
<td>Traditional Structured Analysis</td>
</tr>
<tr>
<td>Software Performance Specification, General</td>
<td>Unified Modeling Language (UML)</td>
</tr>
<tr>
<td>Software Performance Specification, Database</td>
<td>IDEF-1X</td>
</tr>
<tr>
<td>Software Performance Specification, DoD IS</td>
<td>DoDAF</td>
</tr>
</tbody>
</table>
1.2.3 Map models to templates such that for any of the specifications listed above there is a model identified for each paragraph of each specification. Ideally, all of the paragraphs of a particular kind of specification would employ models from the same family as suggested in the pairing above.

1.2.4 Map specialty engineering disciplines to specification templates in preparation for program mapping of these disciplines to specific product entities as a means of directing specialty engineering requirements analysis work.

1.2.5 For each template and model combination, prepare a DID communicating how the analyst will prepare the specification using a particular modeling approach and template. The DID must clearly show the connection between the modeling artifacts and the template paragraphing structure. The DID paragraphing structure must coordinate with the modeling components that are intended to yield derived requirements.

1.2.6 Map the functional departments that will be responsible for performing requirements analysis on programs to the template paragraphing structure for each kind of specification telling where the programs will acquire the analysts to accomplish the specification development work.

1.2.7 Prepare a template and DID for a system definition document (SDD) within which program structured analysis work products will be captured and configuration managed. These work products are to be captured in appendices of the document. An alternative is to capture the modeling artifacts within a computer tool that can be configuration managed.

1.2.8 Map the appendices of the SDD to the paragraphing structure of the templates telling in what appendix the corresponding work products will be captured.

1.2.9 Combine the functional department map (1.2.5), models map (1.2.3), and SDD appendix map (1.2.7) on a single matrix for each template and make these matrices available for program use.

1.2.10 EIT and the functional system engineering department must cooperate on selection of one or more computer tools or paper and pencil algorithms with which to accomplish requirements analysis on programs. Built a generic schema coordinated with preferred methods and models.

1.3 Proposal Work That Prepares the Program for Initiation

Proposal team work is illustrated in Figure 3. Blocks that do not have numbers coordinating them with the steps in this algorithm are not covered by the algorithm because they are not directly related to requirements analysis and specification development but these blocks add valuable context.
When beginning the proposal or program work, the manager should establish a program integration team (PIT) staffed by engineering, manufacturing, verification, logistics, and quality and a program business team (PBT) staffed by finance, contracts, scheduling, business information systems, and administration. Both of these teams should report to the program manager. These two teams could be combined as a staff function to the proposal manager but they will have integrating and optimization roles to play across the product oriented teams.

The PIT will perform an initial system analysis that will result in a clear understanding of any requirements provided by the customer, formatting of those requirements into alignment with the enterprise DID for a system specification, and adding to those requirements the results of their own system analysis work. A system environmental requirements analysis activity will expose a set of tailored standards covering the natural environment corresponding with spaces within which the system shall have to operate. A threat analysis will lead to exposure of hostile requirements. An interface analysis will identify external and top level internal interfaces that will be characterized in requirements statements. The result will be a preliminary system specification for submission with a proposal. If possible, this analysis work will be continued to develop all of the immediately subordinate specifications each of which will be the development responsibility of one of the top level integrated product and process teams (IPPT) to be identified and staffed subsequent to contract award.

The modeling work described in paragraph 1.3.2 will yield modeling artifacts from which requirements may be derived. The preliminary system specification development and any other specifications developed during the proposal effort will follow the pattern described in the
next paragraph. The second tier specification development may be delayed until a contract award but should precede the formation of the top tier IPPT. In all cases, requirements are derived from a model.

1.3.4 Requirements flowing from the structured analysis work will flow into a requirements analysis sheet (RAS) implemented in a computer database tool. All requirements entered into the RAS must include a traceability reference to the model from which they were derived.

1.3.5 Out of the initial PIT system analysis work will also come the preferred product entity breakdown diagram upon which the PIT, working in concert with integrated business team personnel, will construct overlays for organization responsibility breakdown (IPPT assignments), specification tree breakdown, engineering drawing breakdown, work breakdown (WBS), and manufacturing breakdown. The work breakdown will be handed off to the business team that will use it as the basis for building the program work definition, cost estimate, and IPPT work budgets. The IPPT will be assigned so as to align perfectly with the WBS making it possible to present to each IPPT leader as the teams are formed, a copy of the top level specification for which the team is responsible and the related budget and planning package for the whole WBS the team is responsible for as well as their top level schedule responsibilities encouraging the result that the IPPT leader may be held accountable for managing all aspects of the development of the entity assigned to the team.

1.3.6 The PIT will select a set of templates that correspond with the kinds of specifications that will have to be prepared on the program and the related DIDs that are coordinated with the models that will be applied in the analytical work. The PIT must also map specialty engineering disciplines to product entities to aid teams being formed to staff appropriately.

1.3.7 The PIT must take action to cause adequate computer tool seats to be allocated to the proposal team and accomplish any planning necessary for the subsequent program relative to the use of any requirements database tools and make any needed adjustments in database schema for the program.

1.3.8 The PIT shall capture the results of structured analysis work performed during the proposal activity in a preliminary system definition document (SDD) that will be used as the basis for subsequent lower tier analyses.

1.3.9 Any specifications developed in the proposal effort must be formally reviewed and approved by the proposal manager.

1.4 Work Subsequent to Contract Award

Specification related work to be accomplished subsequent to contract award is illustrated in Figure 4. This work is repetitious in nature progressively working down through the expanding architecture. Top-level teams may shred out during program work yielding sub-teams but in all cases, the top-level teams are responsible as managers for all lower tier team activity. This telescoping management responsibility is applicable throughout the team structures. During program performance, lower tier requirements analysis responsibility may be passed down
through the team structure with immediately superior team reviewing and approving of all immediately lower tier team specifications or the responsibility may be retained by the higher-level team but these decisions must be coordinated with the team budgets and staffing considerations arrived at during proposal work.

Figure 4  Program Work

1.5  The Preferred Templates

Ideally, the development organization would build a set of templates (paragraphing structure with no content) and data item descriptions (DID) that tell how to build a specification following the related template using a particular set of models. These should be maintained by the functional department in system engineering that has overall requirements and specification work responsibility. These should be available for reference or download by any new program.

Figure 5 offers a view of the preferred template for a system or hardware specification using traditional structured analysis as the modeling choice. The template is annotated with the preferred modeling artifact that will be used to identify the corresponding requirements and the functional department from which the program will obtain personnel to accomplish the related modeling work using the Figure 3 organizational structure. The data item description (DID) acronym in the model column means that the content is driven by the content of the DID used as the basis for the specification. The department references are cut at a very high level in this case and should be identified at one or two layers below this level but Figure 3 goes no deeper. The APP column gives the Appendix in the System Definition Document where the work products can be found.

The structure in Figure 5 can also be used for computer software requirements specifications (SRS) with paragraph 3.1 rewritten for UML and the model column updated to reflect UML artifacts.
<table>
<thead>
<tr>
<th>PARA</th>
<th>TITLE</th>
<th>MODEL</th>
<th>DEPT</th>
<th>APP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>SCOPE</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>2.</td>
<td>APPLICABLE DOCUMENTS</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.</td>
<td>REQUIREMENTS</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1</td>
<td>Functional and performance requirements</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.1</td>
<td>Missions</td>
<td>Mission Analysis</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.2</td>
<td>Threat</td>
<td>Threat Analysis</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.3</td>
<td>Required states and modes</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.3.1</td>
<td>Functional analysis</td>
<td>DID</td>
<td>2100</td>
<td>A</td>
</tr>
<tr>
<td>3.1.3.2</td>
<td>Subordinate entities</td>
<td>DID</td>
<td>2100</td>
<td>C</td>
</tr>
<tr>
<td>3.1.3.3</td>
<td>Interface relationships</td>
<td>DID</td>
<td>2100</td>
<td>D</td>
</tr>
<tr>
<td>3.1.3.4</td>
<td>Specialty engineering requirements</td>
<td>DID</td>
<td>2100</td>
<td>E</td>
</tr>
<tr>
<td>3.1.3.5</td>
<td>Environmental model</td>
<td>DID</td>
<td>2100</td>
<td>B</td>
</tr>
<tr>
<td>3.2</td>
<td>Entity capability requirements</td>
<td>Functional Analysis</td>
<td>2100</td>
<td>A</td>
</tr>
<tr>
<td>3.2.m</td>
<td>Capability m</td>
<td>Functional Analysis</td>
<td>2100</td>
<td>A</td>
</tr>
<tr>
<td>3.2.m.n</td>
<td>Capability m, requirement n</td>
<td>Functional Analysis</td>
<td>2100</td>
<td>A</td>
</tr>
<tr>
<td>3.3</td>
<td>Interface requirements</td>
<td>N-Square Diagram</td>
<td>2100</td>
<td>D</td>
</tr>
<tr>
<td>3.3.1</td>
<td>External interface requirements</td>
<td>N-Square Diagram</td>
<td>2100</td>
<td>D</td>
</tr>
<tr>
<td>3.3.1.m</td>
<td>External interface m</td>
<td>N-Square Diagram</td>
<td>2100</td>
<td>D</td>
</tr>
<tr>
<td>3.3.2</td>
<td>Internal interface requirements</td>
<td>N-Square Diagram</td>
<td>2100</td>
<td>D</td>
</tr>
<tr>
<td>3.4</td>
<td>Specialty engineering requirements</td>
<td>DID</td>
<td>2100</td>
<td>E</td>
</tr>
<tr>
<td>3.4.m</td>
<td>Specialty Engineering Discipline m</td>
<td>Specialty Scoping</td>
<td>E</td>
<td></td>
</tr>
<tr>
<td>3.4.m.n</td>
<td>Specialty Engineering Discipline m,</td>
<td>Specialty Scoping</td>
<td>E</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Requirement n</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.5</td>
<td>Environmental conditions</td>
<td>3-Layered Env Model</td>
<td>2100</td>
<td>B</td>
</tr>
<tr>
<td>3.6</td>
<td>Precedence and criticality of requirements</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>4.</td>
<td>VERIFICATION</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>PACKAGING</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>6.</td>
<td>NOTES</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>6.1</td>
<td>Requirements traceability</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>6.1.1</td>
<td>Inter-specification specification traceability</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>6.1.2</td>
<td>Verification traceability</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>6.1.3</td>
<td>Modeling traceability</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>6.1.4</td>
<td>Section 2 traceability</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>6.1.5</td>
<td>Programmatic traceability</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>6.2</td>
<td>Glossary</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>6.3</td>
<td>Specification maturity tracking</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>A.</td>
<td>APPENDIX A</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
</tbody>
</table>

Figure 5  System and Hardware Specification Template

A similar mapping should be provided for the software requirements specification (SRS) based on, in the author's view, the use of UML. Customers often require conformance to a DID supplied by them but may permit tailoring. The outline included in Figure 5 is a significantly tailored version of MIL-STD-961E to group all requirements so as to correspond with the modeling components contained in traditional structured analysis described in paragraph 2.2.1. k of the standard. The author believes this format will work with software as well but a different DID is required coordinated with the modeling approach selected (UML encouraged).
1.6 Modeling Work Product Capture Document

Programs should also be provided with a template for a System Definition Document (by whatever name) within which they can easily capture the results of all modeling work so that it can be preserved beyond the period of time when that work is actively being pursued. A later section describes this document. The appendices of the SDD are referred to in Figure 5 in the APP column and explained in Figure 20 and related text.

2 Structured Analysis

There are many models that can be used to accomplish an organized requirements identification effort that is preferred to an ad hoc method because it will most often hit the target noted earlier - identification of all of the essential characteristics and identification of no unessential characteristics. This process description encourages the use of traditional structured analysis in the near term to develop and identify the requirements for systems, hardware, facilities, real property improvements, and personnel actions captured in procedures as discussed in paragraph 2.1. One of the most difficult tasks in system development revolves around the relations between the system entities, the interfaces. This discussion of traditional structured analysis also contains a complete algorithm for identifying all system relations. Where the product is going to be implemented in computer software, unified modeling language (UML) is encouraged within a process context offered in paragraph 2.2. It is intended that a development organization should develop a process transformation roadmap needed to transition from this mixed method for the modeling work to a universal modeling approach, still evolving, that can be applied to all requirements work at the earliest possible. See paragraph 2.3 for a discussion of integrated modeling.

2.1 Traditional Structured Analysis

Figure 6 illustrates an overview of the traditional structured analysis process (TSA). The eleven numbered steps are briefly introduced followed by additional details in subordinate paragraphs.

1. Understand the User Requirements - Through conversation with the user and/or reading a user requirements statement or specification, the developer tries to understand the user's need. This is not ever easy because the user is able to explain only what their mission interests are and the developer needs hard engineering data.
2. Decompose - Users commonly have problems that are too grand to be easily understood in a single small document or simple diagram. These problems commonly have to be decomposed or partitioned into a collection of smaller related problems.
3. Functional Flow Diagram - The TSA approach employs some form of functional analysis as the decomposition medium.
4. Performance Requirements Analysis - The functions are translated into performance requirements.
5. Requirements Analysis Sheet (RAS) - The strings of functions, performance requirements, and product entities to which they are allocated appear as rows in the RAS. The RAS also is used to capture all system requirements linked to the model from which they were derived.
6. Requirements Allocation - Performance requirements are allocated to product entities in the RAS.
7. Product Entity Structure - The physical and logical entities that comprise the system are arranged in a hierarchical structure that is used as the basis for WBS, specification identification, team assignments, and many other program applications.
8. N-Square Diagram - The interfaces that must exist between the product entities are identified through a pair-wise analysis of all possible interface relationships.
9. Environmental requirements for the expanding product entities are determined through application of a three-layer model.
10. Specialty engineering requirements are identified by a group of specialist linked to the product entity structure by a specialty engineering matrix.
11. This process is applied iteratively as lower tier entities are identified through functional analysis.

Figure 6  Overview of the Traditional Structured Analysis Process
2.1.1 A System Defined

A man-made system is a collection of entities that are meant to interact in predictable ways with an environment and with each other via relations between them to achieve a useful function identified and articulated by a customer as a mission need statement. Therefore, systems are composed of entities and relations between the entities. The system is intended to satisfy the mission need statement, the system’s ultimate function, depicted on system diagrams as a rectangular block titled System Need and identified with a functional identifier F. The need is allocated to the system depicted on system entity model by a rectangular block named “system” (or a particular system) and identified with a product entity identifier A.

A system interacts within an environment as shown in Figure 7. The environment for every system is everything in the Universe (U) less the entities that are part of the system product entity structure (Q = U - A). One can reduce the scope of the environment to those elements that will have some influence on the system. The line joining the system and environment in Figure 7 (I2) indicates the relations between the two (external interfaces). The line joining the system on both terminals (I1) indicates the internal relations between system entities (internal interfaces) yet to be defined within the system.

![Figure 7 Ultimate System Diagram](image)

2.1.2 The System Environment

The environment for any system is composed of the subsets illustrated in Figure 8. While all environmental effects on the system are relations, they may be partitioned between those that are commonly considered environmental stresses and the cooperative environmental elements that are treated as external interfaces commonly developed by a pair of teams or contractors responsible for the terminal product entities.

A context diagram, such as that shown in Figure 9, even though similar in nature to Figure 7, offers a useful simple model for focusing attention on identifying all external relations. Some of these terminators will be natural, non-cooperative, or induced environmental stresses. Others include hostile stresses determined through a system threat analysis as well as both stresses and useful relations with cooperative systems. Though this diagram was conceived as the beginning of the modern structured analysis model, it has a useful purpose in TSA as a means of viewing all of the inside-outside relationships and in UML as an organizer of use cases. It can be used to make a first impression in the line I2 in Figure 7.
The system natural environment is determined by defining all of the spaces within which the system will be employed based on an analysis of the intended mission and basing concept. The spaces are coordinated with a set of environmental standards. Each standard is studied for necessary content and the remainder tailored out. Each selected parameter is then studied for an appropriate range. The system natural environment is then the union of the selected parameters from the selected standards.

The non-cooperative environment is defined by determining what stresses will be applied to the system from man made systems which are neither hostile nor cooperative. An example of non-cooperative stress is electromagnetic energy. Self-induced environmental stresses are not easily
determined at the system level because one needs to understand energy sources and other stressors within the system determined as part of the design of end items.

System cooperative environmental relations are defined by determining how the system to be developed will associate with other friendly systems already in existence or being developed. These associations may be coupled into or out of the system in terms of information, physical association, materials, or energy.

2.1.3 System Functionality

A function is a necessary activity for a system to perform. It may be static, dynamic, or both. It should be named using an action verb followed by a noun. A function is depicted in modeling the system as a rectangular block identified by an action verb name centered in the box and a function identifier (ID) in the lower right corner. The ultimate function for any system is the customer need the name of which is the need statement possibly paraphrased to fit into the space provided coupled with a function identifier $F$.

Two or more functions can be linked together using directed line segments to show a sequence of functions. In Figure 10 the understanding is that function $F_1$ must be accomplished before function $F_2$. Combinatorial symbols may be added to permit more complex sequential relationships. The combinatorial symbols encouraged are AND, inclusive OR (IOR), and exclusive OR (XOR) with the common logical meanings. Enhanced functional flow block diagramming adds loop (repeat a function until a specific outcome has been attained) and iterate (repeat a function a specific number of times) combinatorial symbols that can be useful. Diagrams so constructed are called functional flow diagrams. These diagrams may be oriented on the page with their primary flow axis arranged horizontally or vertically with the flow in either direction.

![Figure 10 Function Sequence](image)

For any function with identifier $F@$ (where $@$ is a string of length $n$ (including $n=0$) composed of characters from the set \{A through Z less O\}U(a through z less l)U(0 through 9)) there may exist one or more subordinate functions $F@#$ (where $#$ is a single character from the same set identified above which differentiates other functions at that level from one another). This is illustrated in Figure 11. Every function need not have an expansion. There is no need to assign function identifiers in alpha numeric sequence on a page of the diagram but it helps the human to use the diagram if they are assigned initially coordinated as much as possible with the function sequence. If a function is deleted subsequent to a release of the diagram, that identifier should not be used again. If the number of functions on any one diagram exceeds the maximum number of symbols available, 60, change the diagram to reduce the number to less than 60.
Ideally, whoever accomplishes the initial analysis of the need, would do so using the functional analysis process described here where the first decomposition is the system life cycle as shown in Figure 12 and the second is an expansion of the life cycle function “Use System” (F47) to expose the top level operational intent and initial content of the user requirements documentation or preliminary system specification. If the customer or other initial analysis agent applies an unstructured or ad hoc approach, then the development organization may have to accomplish a functional analysis and try to map the requirements identified by the customer (user and acquisition agent) to the functionality exposed when they do accomplish this work.
Ideally, the development organization would extend the functional analysis into the remainder of the system life cycle functions as well as the Use System function determining appropriate resources for the process steps of the development program and the product system being developed such that the physical product delivered will be jointly optimized relative to its product and process. Commonly, process functions do, or should, influence product functions and the corresponding product entities needed as well as the opposite case.

2.1.4 Performance Requirements Derivation and Allocation

The functions identified in the functional flow diagram must be translated into performance requirements that tell what the system and its parts must do and how well it must do them as shown in Figure 13. These statements can be first developed as primitive statements for example phrased as “Velocity ≥ 600 knots” without complete sentence structures and subsequently transformed into complete sentences in the chosen language. Traditionally, a requirements analysis sheet (RAS) has been used to capture the function identification, the primitive performance requirements statements, and the allocation of these performance requirements to product entities. One could allocate the function names directly to architecture but often one finds a one-to-many allocation result this way whereas allocation of performance requirements tends to follow a one-to-one pattern. To fully characterize a function it may require identification of multiple performance requirements and these several performance requirements may be allocated to different product entities.

The RAS as traditionally used is incomplete, unfortunately, and this discussion will use a RAS complete. We will show how the three kinds of constraints can also be captured in the context of Figure 13 shortly. The intent is to be able to capture all requirements in a RAS linked to a modeling artifact implemented in a computer application.

<table>
<thead>
<tr>
<th>FUNCTION</th>
<th>PRODUCT ENTITY</th>
<th>REQUIREMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>MID</td>
<td>NAME</td>
<td>PID</td>
</tr>
<tr>
<td>F123</td>
<td>Provide Thrust</td>
<td>A12</td>
</tr>
</tbody>
</table>

Figure 13 Traditional Requirements Analysis Sheet

2.1.5 Product Entity Structure

System functionality is accomplished by physical entities that form part of the physical system. These entities in the system, in the aggregate, comprise the product entity structure that the author formerly referred to as the system architecture. The word architecture has taken on a significantly broader meaning in recent years convincing the author that what he formally referred to as architecture should now be referred to by the more isolating term product entity
structure. The function F is accomplished by product entity structure element A, the system, by definition as shown in Figure 7. Lower tier entities should be exposed following Sullivan’s encouragement of form follows function. Trade studies may be appropriate to make hard decisions on the best implementation of a particular function in early program phases. The physical entities that accomplish functionality can be partitioned into five classes: (1) hardware, (2) computer software, (3) facilities and improvements to real property, (4) procedural definition, and (5) humans following procedures or acting autonomously. We could merge the last two into one. The relationship between functions, the physical entities in the product entity structure, and the corresponding performance requirements is depicted on a simple requirements analysis sheet as shown in Figure 13.

The aggregate product entity structure for a system is illustrated in a hierarchical model connecting the product entities arranged into a breakdown block diagram as illustrated in Figure 14. The entity identifiers follow the same pattern as the function identifiers explained earlier beginning with a letter A because of the author's previous use of the word architecture for this view. The entities stream out of the RAS available for structuring into the product entity block diagram. Ideally, this work would be accomplished by a team of people representing hardware and software engineering, manufacturing, procurement, and verification with strong system/program leadership by the PIT. Initial functional analysis and allocation must concentrate on understanding user mission needs. This will generally require intense interaction with the user, ideally using system models to encourage mutual understanding. Alternative ways of implementing functionality with different product entities should be considered and where the decision is very difficult they should be subjected to a trade study.

Figure 14 Product Entity Structure
2.1.6 Allocation Pacing Alternatives

The conduct of the functional analysis and allocation work can be paced in one of four different ways:

1. Instant – As soon as functions and/or their corresponding performance requirements are identified, they must be immediately allocated to the expanding product entity structure.

2. Terminal - All of the functional analysis must be complete before any functions and/or their corresponding performance requirements can be allocated to product entities.

3. Layered - The analyst completes one layer of the expansion of a function and must allocate all of the exposed functionality and/or the corresponding performance requirements to product entities before further expanding that function. This works best if all product entities related to a layer are defined in terms of their requirements and design concept before pursuing the next functional layer and its allocation but there generally is not sufficient time available and one must accomplish a lot of this work in parallel leading to some risk and interaction. The layered approach has been popularized by Mr. Bernard Morais and the late Dr. Brian Mar under the title FRAT.

4. Progressive – The analyst completes several layers of the functional analysis without allocating any of it to product entities. At a layer guided by experience the analyst begins allocating performance requirements derived from higher tier functionality to product entities. Design concepts are defined for the product entities at the higher levels and these act to both guide and constrain lower tier functionality progressively. Allocation is delayed throughout the analysis such that higher tier design concepts help to steer lower tier functional analysis tending to isolate iteration to one structure (functionality, architecture, or allocation) at a time. The two extreme cases (1 and 2) are flawed due to a need to iterate F, A, and allocations excessively in the case of 1 and a common need to significantly change lower tier functionality after the higher tier allocations begin in the case of 2. The progressive approaches either by layer or guided by experience will generally produce better results with less modeling hysteresis.

There exists a downward limit in decomposition or expansion of the functional portrayal. This limit for any one branch in the expansion is best determined on the product entity plane based on the analyst’s understanding of the problems related to the lowest tier product entities. Where all of the lowest tier entities are fully characterized supporting procurement or in-house design, the functional analysis work can be reduced to maintenance of the models and related data. There is one more layer of requirements related to parts, materials, and processes but that is driven by design decisions during synthesis and commonly has already been completed by those responsible for the sources of these entities. The logistics analysis process may require a switch to a process diagram and if the progressive allocation approach described has been followed the functional flow will have been migrated toward a process diagram at the lower tiers.
Figure 15 illustrates a geometric view of the process of deriving performance requirements from functions and the allocation of those performance requirements to product entities. For example, a performance requirement has been derived from F4713 and allocated to product entity A11. This plane represents the normal RAS used only for function allocations. As suggested by the other structures we will use the geometric structure to expand the RAS to cover all requirements.

Figure 15  Juxtaposition of RAS and N-square Diagrams

2.1.7  System Relations

As the product entity structure is formed, the analyst can begin to identify the needed internal relations between the system entities by using an n-square diagram where the product entities are identified down the diagonal at some level of indenture. For a given analysis where the number of entities being studied for interface relations in an n-square diagram is N the largest possible number of relations is N(N-1) counting each direction as one possibility between each pair of entities. Interfaces between these entities is pre-determined by the way that functionality is allocated to the entities. Therefore, one may explore the list of function-product entity pairs
associated with each product entity in the n-square diagram. This is referred to as a pair-wise analysis of the n-square diagram intersections.

Figure 15 includes an n-square diagram with only half of the square showing (the remainder hiding behind the other structures in the diagram). The diagonal (containing the product entities we are interested in accomplishing the pair-wise analysis on) has been aligned with the product entity axis of the function-product entity matrix (which corresponds to the simple RAS of Figure 13).

The process for marking the intersections of the function driven relations matrix (n-square plane) entails a pair-wise analysis of the function-product entity pairing relationships marked on this matrix. Interface Ix is encouraged by the conclusion that if F4711 maps to A13 and F4712 maps to A11 then there is a possible demand for an interface between A11 and A13 to implement that relationship. So we map functions to product entities but we map F-A pairs to interfaces and those interfaces are pre-determined by the way we allocate functions to the product entities.

If the system is a modern war ship or the system that will take man to Mars, a pair-wise analysis of the function driven relations matrix would be beyond human comprehension if attempted all at one time, We can, however, partition the analysis work to one interface expansion at a time and it is not so overwhelming. At any one level of product entity granularity, we can explore one layer of product entity structure expansion for internal interfaces within the parent item. If there are five subordinate entities then the number of possible interfaces to be examined in a pair-wise fashion would be 5x4 or 20. Similarly, external interfaces can be analyzed individually. Of course, it will still be necessary to accomplish considerable interface integration work because of the partitioned process. This process will go very fast if the analyst is very familiar with the problem space and the evolving solution concepts. The complete algorithm is extended in subsequent paragraphs.

The requirements analysis sheet (RAS) identifies every possible pairing of functions and architecture entities. We may sort this listing so that all of the functions (or performance requirements derived from those functions) allocated to each entity are grouped by entity. Then we can pile up the allocations onto the product entity squares on the diagonal of a physical n-square diagram as suggested in Figure 15.

The discussion so far has dealt only with internal interface identification all defined on the function driven relations matrix of Figure 15. To cover external interfaces we add the larger n-square diagram on Figure 15. The diagonal in this case includes all of the product entities plus all of the external entities in the cooperative environment. We can identify relations between these external and internal entities in the same way we did the internal ones. The association of function F4712 to cooperative environment entity QC1 and allocation of a function to A12 may define a need for an interface I2y.
2.1.8 Environmental Relations Algorithm

2.1.8.1 System Environmental Relations

The system environment consists of all entities in the Universe less those that are in the system. That is, \( Q = (U - A) \) where \( Q \) is the environment, \( U \) is the Universe, and \( A \) is the product entity structure of the system being developed. It is convenient to partition all system level environmental relations into the sets illustrated in Figure 8. The system cooperative environment (QC) can actually be treated as an external interface and can be developed using the algorithm covered in Paragraph 2.1.7 very effectively. External cooperative systems are simply located on an extension of the product entity axis forming the cooperative environment axis. The hostile environment (QH) can best be understood through analysis of threats posed by hostile forces. The non-cooperative environment will yield to the same thought process applied in the threat analysis except that the stresses applied to the system are not applied for hostile purposes, rather simply because the system being developed is sharing a common operating space with other systems. Electromagnetic interference is an example of the stresses applied in this set.

System time (QN2) is studied using time lines oriented about the functions that the system must satisfy. When we allocate those function (or their corresponding performance requirements) to architectural entities the timing requirements corresponding to the functions are applied to the entity as timing requirements.

System space (QN1) is defined through mission analysis such that it is determined in what volumetric spaces of the Earth (surface, subsurface, and aerodynamic) and/or surrounding space and/or distant bodies the system shall function within, on, or around. For each space, that space is teamed up with one or more natural environmental standards that define that space. Each of these standards is then studied to determine which natural environmental (QN3) parameters included in the standard shall apply to the system being developed. Those that do not apply are tailored out of the standard. The selected parameters are then studied to ensure that the range of values is appropriate for the system. If the range in the standard is too broad it is tailored to narrow the range of values to that for which the system shall be designed. This process is repeated for each standard linked to a system operating space.

Figure 15 also extends the RAS concept to include environmental requirements analysis. The system environment as depicted in Figure 7 is illustrated at the diagram “origin” on the environmental axis that happens to coincide with the architecture “origin” that corresponds to the whole physical system A.

2.1.8.2 End Item Service Use Profile

An end item is a major element of a system that generally retains its physical configuration throughout its mission performance and has an end use function. The end items may remain fixed in place during use or move over great distances and maneuver within the system spaces as a function of the system mission and the end item’s application in the system. Each end item should be designed to endure only those natural environmental stresses anticipated so it is necessary to determine what subset of the system natural environment each end item shall be
exposed to. To accomplish this, one must create a physical process flow diagram. This is not the same as the functional flow diagram used to identify system architecture and performance requirements. The blocks on a functional flow diagram represent things that have to happen whereas the block of a process diagram represent a real world analogy. You cannot profitably consider system entities flowing through the functional flow diagram, indeed we are using the diagram to determine what those entities should be. However, we can imagine the system product entities flowing in the process diagram where each process acts as a transformation on the system entities.

The first step in defining end item environmental relations is to map the system environmental parameters to the process steps at some level of indenture, generally at a level where the environmental map does not change significantly below that process level. This work defines an environment for each process. The next step is to map the architectural entities to the process blocks. If an entity only maps to a single process step, it simply inherits the process environmental set. If, as is so often the case, the architectural entity maps to two or more process blocks, then it will be necessary to apply some kind of integrating process to any differences in environmental stresses and their values observed between the two or more process blocks. The rule most often selected initially is to pick the worst-case range for each parameter across the process values being evaluated. If this rule does not adversely influence system cost, then it is an adequate solution. If this approach either results in an adversely narrowed system solution space, then an alternative to “worst case” must be derived. Often time lines will show that the problem environmental stress will be applied over such a short time as to be insignificant. In other cases, one may find that the problem can be narrowed to some particular combinations of values that are very unlikely to occur. If the problem is intractable, it may be necessary to restrict one or more system environmental parameters more severely than is currently being done. Generally, this will have an adverse effect on system performance.

The self induced environment (QI) can best be studied and defined at the end item level since it is end items which contain the sources of energy and other stresses of interest which will reflect commonly through a natural environmental parameter right back into the system. Since the self-induced stresses are commonly greater in magnitude than the corresponding natural stresses for that same parameter, these induced relation values have the effect of extending the range defined through the application of the end item service use profile algorithm discussed above.

2.1.8.3 Component Environmental Relations

The environmental relations appropriate for components installed within end items are simply the end item environmental stresses if the end item has no altering effect on those stresses and all spaces within the end item offer the same environmental stresses to components installed within them. Where an end item does have a modifying effect on end item stresses but all spaces within and upon the end item offer the same stresses, it is necessary to determine the end item design effect on end item environmental stresses and the result is the set of installed component environmental stresses. The most complex case occurs when the end item must be partitioned into two or more spaces more often called zones of common environmental stresses. The value of each end item parameter must be determined for each zone thus defining the environment for each zone. Then one maps the components to the zones and they inherit the zone environments.
Commonly, the job is not complete at this point because it is found that there is no zone within which one or more components can be installed in a particular end item that will cause their environmental stress limits to be satisfied. When this happens, it is necessary to either change the component environmental specification values or include an environmental control system as an added entity into which the components with the environmental range shortfall problem are placed.

2.1.8.4 Environmental Impact

The environmental effects discussed in the three previous cases deal with environmental stresses the environment will apply to the system but there may be cases of the opposite direction that will cause damage to the environment. Once identified by someone skilled in environmental impact, these can be treated like safety hazards to be mitigated through re-design, procedural controls, or compensating environmental actions. In the case of military systems it is very difficult to mitigate the damaging effects of warfare but these systems can also have damaging effects on the environment in peaceful use such as training and maintenance. Often these materials are just naturally dangerous to be around as illustrated in the difficulties observed in the base closure efforts where many adverse environmental effects have had to be identified and mitigated.

2.1.9 Specialty Engineering and RAS Complete

The system engineering agent for the system must build a list of all of the specialty engineering disciplines that will be applied in the development of the system. A specialty engineering scoping matrix should be prepared between specialty engineering disciplines that may be required in development of the system and the product entities. This will help to determine team staffing needs in that area and connect people in those disciplines with a need to do specialty engineering requirements analysis for the indicated items. Figure 16 adds one more plane, the specialty engineering scoping matrix, to the construct previously illustrated in Figure 15 providing for allocation of specialty engineering disciplines listed to architecture.

Specialty discipline H7 is shown mapped to architecture item A11. This must be followed by analyst definition of one or more discipline H7 requirements that will flow into the specification corresponding to the product entity. The structure exposed in Figure 16 is a complete RAS showing all of the important requirements related relationships supporting the requirements analysis process leading to the identification of every kind of requirement appropriate to the system and hardware specifications and all of them linked to and derived from a model.
The fact that an aircraft airframe will have to be checked for alignment during manufacture and after hard use (hard landing or pulling excessive g’s in flight, for example) identifies a need for a relation between the airframe and the equipment which will be used to accomplish the alignment. Today this will generally call for some form of laser optical application so the airframe would have to either include targets, detectors, or mirrors or provisions for these to be applied. The manufacturing and maintenance engineers would have to consider all of the ways there may have to be relations between support equipment and the operational entity. There may be other cases where these disciplines have to call for internal relations within the system entities. For example, if the system must include built in test (BIT), there will have to be relations between most of the on board equipment and some entity that concentrates the BIT effects for display to operations and/or maintenance personnel.
The needs of operations personnel, such as aircraft pilots, locomotive engineers, ship captains, and automobile drivers provide a tremendously rich class of entity relation possibilities. The physical relations are always fairly simple in that the human operator has only his/her senses, mental acuity, and physical strength through which to interact with the system. This problem is made much more complex, however, because not all humans will react in precisely the same way to a particular stimulus. It will be necessary to determine the complete data set that the human will require under all operating conditions and in what way the human shall influence system behavior in terms of controls. Operator sequence diagrams, built like a UML activity diagram with swim lanes, can be useful in doing this work.

Every one of the specialty engineering disciplines selected for the program must be evaluated for entity relation driving potential and those persons doing that work alerted to their responsibilities in identifying relations for further consideration by the whole team.

2.1.10 RAS-Complete in Table Form

The results from the analyses noted in prior text must be captured on its way into program specifications. Certainly, the most advantageous way to do that is in a computer database systems such as DOORS, CORE, or SLATE. Therefore we would expect that some form of tabular structure would suffice. Figure 17 is offered as the candidate view of this table for use during development in capturing the relationship between model, requirements, product, and document entities. The model ID (MID identifies the model from which the requirement was derived. The requirement columns identify the requirement ID (RID) assigned by the computer system for use in traceability. The RID is a made-up computer-assigned unique field using a base 60 numbering system in this example but a hexa-decimal implementation is probably more common. Ideally the requirement statement should be captured in primitive form (attribute, relation, value, and units) wherever possible with different fields for each component of the string. The primitive form is shown concatenated in Figure 17. The final column pair offers specification paragraph number and title.

The sample data included is ordered by model ID alphanumerically separating the data into the four kinds of requirements found in a system or hardware specification. The lists the MID respected by the author is still in a state of change as different RAS-Complete possibilities are explored. This may explain the apparent unthinking selection of H and Q for specialty engineering and environmental requirements, respectively. The intent is to identify all possible modeling artifacts with a letter as a source from which every conceivable requirement may be derived. Model letters for UML artifacts have been included in the set and will be introduced later.

This view provides clear traceability between the models from which the requirements were derived and the product entities to which they were allocated for all of the requirements, not just the performance requirements. What the author calls lateral traceability is captured in the database implementing the RAS-Complete. It is also a simple matter to link the rows in the matrix in a database to the corresponding verification requirements as well as the tasks to which they are allocated and their corresponding plans, procedures, and reports. Vertical traceability is, of course, simply a matter of relating the unique RIDs from pairs of requirements in specification parent-child relationships identified by their PID.
2.1.11  Traditional Structured Analysis Summary

In summary, a system is defined by identifying its functionality starting with the need (F), allocating that functionality to entities that become part of the system architecture (A). These entities that form the system architecture are selected by determining the performance requirements that the system must satisfy to meet the top-level customer’s need. The pairs of functions and product entity allocations pre-determine how the entities will have to relate to each other through interfaces (I) between the product entities. The environmental elements (E) are defined at the system level in terms of the spaces within which the system must function and the corresponding characteristics of those spaces drawn from appropriate environmental standards covering those spaces. As depicted in Figure 18, the traditional structured analysis effort attempts to define the most cost effective solution such that in N cycles of the process axis of the physical system (generally cyclical in the interest of reuse of system elements) the relation P (process) maps the cross product of the power sets of architecture (A*), interface (I*), and environment (Q*) to the function set F such that F is covered.
For every process $P_i$, there exists a combination of architecture, interface relations, and environmental stresses such that some subset of the function set is covered or accomplished. The power sets of these entities include all of the possible subsets of these entities within their own set thus the power set of $A$ includes every useful combination of product entities relative to every process step. Useless subsets are also included in the power set as well, of course. It is important that the functions be covered in the correct order determined by the sequence of the processes linked together in the process axis. If all of the functions are satisfied in $N$ revolutions of the process axis as planned, then we may say that the system is consistent relative to the use of its product entities, interfaces, and environmental stresses. If there are product entities that are not used in the process or some that are needed but not available we may not have the optimum product entity structure.

This whole process happens in practice somewhat backwards in that for an unprecedented system, one begins the development process only knowing the ultimate function, the need, and must expand everything from that one perspective.

2.1.12 SDD Content and Format

When used to support the application of traditional structured analysis on a program, the preferred SDD format consists of a main body and seven appendices, each providing a capture point for the work products of one of the several fundamental analytical system requirements analysis process areas. Figure 19 shows how the document is structured. A series of seven interactive system analysis activities feed the development of the appended data explained in subordinate paragraphs. The appended data then becomes the basis for lower tier analysis that produces content for the lower tier specifications and adds to the appended data.
2.1.12.1 Document Main Body

The main body simply contains a table of contents, list of illustrations, and list of tables for the document plus it should provide text explaining the capture of work products in the seven work areas during system and lower tier analyses. The body should also explain that the SDD couples the structured analysis work and its work products to specification content as guided by the selected specification standard templates.

2.1.12.2 Appendix A, Functional Analysis

This appendix captures the functional flow diagram starting with the identification of the system need and the life cycle flow diagram. The Use System Function is initially decomposed progressively to expose more details about the user need. For each block in the functional flow diagram, there should be one line in the function dictionary also contained in Appendix A.

2.1.12.3 Appendix B, System Environment Analysis

The environment consists of several subsets of stresses that are applied to the system. This appendix identifies and characterizes them. Timelines capture critical timing requirements. The spaces within which elements of the system must function are identified and the corresponding environmental stresses defined in terms of standards that describe those spaces. Service use profile analysis is applied to uncover end item environmental requirements. Finally, zoning of end items exposes component environmental requirements.
2.1.12.4 Appendix C, System Product Entity Analysis

The system product entities result from the allocation of functionality to things. As these pairs are defined on the function-product entity matrix, they must be entered into the product entity structure diagram. This work should be accomplished by a team of people knowledgeable in system, hardware, and software engineering, manufacturing engineering, verification engineering, logistics, material and procurement, and logistics in order to evolve an optimum product entity structure which will be universally respected on the program. This product entity structure is also the basis for the specification tree. Each item on the tree must have a responsible agent identified, a template selected, and a release date established. This structure should also be the basis for any IPPT established on the program. It is also the basis for the WBS so the SOW and IMP align perfectly with the teaming structure applied on the program.

2.1.12.5 Appendix D, System Interface Analysis

Interfaces are identified by pair-wise evaluation of function allocations to product entities using an n-square diagram. This appendix identifies all interface needs internal to the system as well as externally to the cooperative systems identified in Appendix B.

2.1.12.6 Appendix E, Specialty Engineering Definition Analysis

Appendix E provides a space in which system engineers can capture their work directed at identifying the specialty engineering disciplines that will have to accomplish work on the various entities in the system product entity structure to define the appropriate requirements and subsequently the needed analyses to confirm that those requirements are being satisfied. A specialty engineering scoping matrix is used to report the results of that analysis.

2.1.12.7 Appendix F, System Process Analysis

Appendix F captures the results of a physical process analysis in the form of a process flow diagram. This is used by logistics engineers to drive out requirements related to training, support equipment, maintenance procedures (tech data content), and spares consumption. It is also needed to complete the environmental use profile study reported upon in Appendix B that drives environmental requirements for end items.

2.1.12.8 Appendix G, Requirements Analysis Sheet

The exposed functions are listed in the Requirements Analysis Sheet (RAS) contained in this appendix. Related performance requirements are defined and allocated to a product entity. These performance requirements have to have a paragraph number assigned, title identified, and they can be outputted into a specification following a particular template. That part of the work can be done inside a requirements database system. Ideally, all of this work would take place within a requirements database tool but some organizations may find it preferable for their purposes to do the traditional structured analysis work using pencil and paper followed by capture of the resulting requirements in a word processor or a computer database tool from which specifications can be generated.
In keeping with the Integrated RAS idea advanced in Paragraph 2.1.10, the RAS is not restricted to performance requirements. Specialty engineering, interface, and environmental requirements can also be included so that every requirement appearing in every specification on a program will transition from the analytical model from which it was derived into a specification via the requirements analysis sheet.

2.1.13 Team Activity During Requirements Work

The PIT is responsible for accomplishing all requirements analysis work focused on developing the system specification and the immediately subordinate specifications that will be the top-level specifications for the top level IPPT. In general, this analysis work will be accomplished using traditional structured analysis following the pattern described in this section. PIT initiates the analysis capturing the work products in the SDD thus initiating that document. Requirements derived from the modeling work are entered into the database application initiating the requirements capture. Any customer requirements documentation is also entered into the database and traceability established between these requirements and those developed from modeling efforts that appear in the system specification. Traceability is continued down to the subordinate specifications to be handed over to the top-level teams when formed.

The PIT identifies all system external interfaces and defines them in the system specification or the beginning of an interface control document. Internal interfaces are also identified and defined for the first tier entities below the system level and the next lower tier in order to complete the internal interface definition for the first tier. All requirements are entered into the requirements database application and traceability entered.

The PIT develops a specialty engineering scoping matrix and maps the needed disciplines to the product entities identified and coordinates the indicated domain experts to derive requirements for the system and top-level end item specifications. The system level environmental requirements are derived from system spaces identified and mapped to corresponding tailored standards.

With the top-level specifications completed, the PIT can bring the top-level IPPT aboard. As those teams assemble and become familiar with their specification and the program planning data prepared by the proposal team, they continue the requirements analysis process relative to the functionality of their product entity. The primary role of the PIT switches to integration and optimization across the IPPT. The teams enter requirements data into the requirements database and maintain traceability. As this process continues, it may become apparent that lower tier teams are required in which case the parent team takes over system responsibilities for them. The parent team in all cases must develop the top-level specification for any new team. Also, at some point, a team will identify an allocation of functionality to computer software and the continued analysis of that entity should switch to UML.
2.2 UML

2.2.1 Entry Analysis and Overview

While it is encouraged that the enterprise apply TSA today as the entry modeling technique, it is entirely possible to initially enter the problem space analysis for a system that will be implemented primarily in software with UML rather than TSA or SysML. The suggested process starts at the top with the problem expressed in the system need and illustrated in a context diagram, borrowed from modern structured analysis, like that shown in Figure 20. The context diagram expresses relationships between the system, represented by the bubble, and terminators, representing outside entities deriving benefits from the system and supplying things needed by the system to function, generally information in a computer software system but more general in a system that will be implemented using a collection of technologies. Figure 20 is an alternative to the traditional depiction of the general system as a block labeled System interacting with a block named Environment.

![Figure 20 Context Diagram](image)

While UML could be the entry modeling approach at the system level following the approach discussed in this section, the discussion to follow is based on the assumption that the problem space will be entered using TSA with a recognition at some point of a need to switch to UML as entities are identified that must be developed in software. Figure 21 illustrates a process for applying UML starting at whatever level in the system the program chooses to start applying it. Generally, this will be some level below the system level based on allocation of higher tier functionality to a software entity.

In a given system, the initial analysis may have identified one or more entities that will be implemented in computer software so for each of these separate entities one should build a context diagram. It should be noted that the context diagram was popularized in modern structured analysis and was not adopted by UML but it is a useful artifact with which to identify the inside-outside relationship between the software entity and the entities external to that software with which it must interact. The context diagram is offered as an intermediary view that will lend some discipline and order to the identification of use cases. We will identify one or more use cases for each terminator and perform a dynamic analysis on each of those use cases.
For each terminator in Figure 20, identify one or more use cases through which the intended functionality will be accomplished identifying the actors deriving benefits from the system to be created. One use case may expand into several extended or included use cases to cover a more complex situation. You will note that the opening gambit has been arranged to provide structure in the identification of needed use cases. Next, for each use case, build one or more scenarios dealing with how that use case interacts with the system. These scenarios can be in text form, a list of events, or some kind of diagrammatic treatment.

Then, express each scenario in the form of an activity and/or sequence diagram cast at the UML entry level initially. The activity diagram can be thought of a functional flow diagram similar to that used by software programmers many years ago or by system engineer in traditional structured analysis. It may be drawn vertically or horizontally as far as the author is concerned. Swim lanes may be overlaid upon the activity diagram each of which corresponds with the lower tier entities (classifiers in UML) that will be responsible for implementing activities within their swim lane. This is a key point in the analysis where the analyst must make lower tier product entity structure decisions that should lead to adding software entities to the product entity structure. Some analysts prefer to think of the two-dimensional artifacts in the diagram as states rather than activities or functions. Simultaneity is not permitted in normal state diagramming but UML permits it in activity diagrams to cover decisions and branching in a way similar to that applied in flow charting.

Alternatively, the analyst can use a sequence diagram to open the exposure of the details of a use case scenario. The entities (classifiers) through which the functionality and behavior are explored are identified on what are called the life-lines drawn as dashed lines below selected physical or
logical classifiers. These lifelines correspond with the swim lanes on the matching activity diagram. Directed line segments connect these life-lines to show passage of messages and relationships between the classifiers. As in the activity diagramming, these life-line decisions may identify entities that already have been identified or they may involve entities not previously identified. In the latter case, the PIT must concur in the model extension and add the new entities to the product entity structure.

Each of these diagrams (activity and sequence) identify a lower tier (white box) view of what entities will be needed to accomplish the exposed functionality and behavior, what will have to happen in the system in order to achieve the intended goals of the use case signified in its name, and offer a detailed view of the order in which these things will occur.

The analyst can allocate top-level functions (activities) to specific entities and arrange the blocks of the activity diagram in corresponding swim lanes linked to these entities. These swim lanes will correspond to nodes, or even higher-level entities, at the system level but in any case we can refer to them in general as physical or logical classifiers within UML. If appropriate, analyze each of these classifiers from a dynamic perspective using some combination of sequence, communication, activity, and/or state diagram. The communication diagram shows the same information as a sequence diagram with an emphasis on the entities rather than the relationships between those entities. A state diagram is useful where there exists some finite number of possible conditions within which the entity can exist and there appear to be understandable rules for the entity to change from one condition (state) to another.

Identify requirements derived from these artifacts and capture them in a program-wide RAS linked to the modeling artifact (MID) that encouraged their identification and the physical or logical classifier, referred to more generally by a product entity ID (PID) in the RAS, that will be responsible for responding to that requirement and in the specification for which it should reside.

2.2.2 The Connection Between Modeling Artifacts, Specification Content, and Product Entities

Figure 22 suggests a hierarchical relationship between the elements of the UML analysis and offers a way of assigning MID. The capabilities in the specification format (template Paragraph 3.2) coordinate with terminators, use cases, extended use cases, and/or scenarios. Figure 22 only shows one terminator expanded but the intent is that for any top level software classifier AX (highest tier SW entities) entered into the product entity structure, one or more terminators would be identified and expanded as shown in the one case shown in Figure 22. The software top-level software classifier, AX, is, of course, a member of the product entity structure (PID) where X is a string of base sixty or decimal delimited base ten numerals. The suggested UML MID stream is identified first with a unique UXh MID (with h = e{1, 2, 3} in the example shown in Figure 22) for each terminator.

The terminator MID can be further decomposed using the MID UXhijk pattern composed from a set of use cases, possible extended use cases, and scenarios for each terminator h. These MID are the entries to place in the RAS for software requirements derived from these artifacts. In general, capabilities will be derived from these artifacts and the requirements subordinate to them will be derived form the dynamic modeling artifacts UXhijk1 through UXhijk4.
Figure 22  Hierarchical Relationship Between UML Dynamic Modeling Artifacts

So finally, the requirements for each capability flow out of applying the subordinate UML dynamic modeling artifacts. As in TSA with lower tier product entities identified from lower tier function allocation, the lower tier software product entities flow from the sequence (life-lines) and/or activity (swim lanes) diagramming. This is a particularly satisfying coordination between lower tier entities being exposed through functional analysis in TSA and activity analysis in UML where both are using essentially the same model to identify lower tier entities.

We can continue to apply UML in the lower tiers as covered in Figure 21 treating each classifier as a system in accordance with the steps discussed above progressively identifying nodes, components (possibly in more than one layer), and classes. If the system level problem space was entered using UML as this process moves forward and downward, it will be decided that some classifiers will be developed as software and hardware entities, with the latter possibly splitting in lower tiers into hardware and software entities. The continuing analysis of hardware entities can be accomplished using traditional structured analysis or SysML model artifacts while the modeling of software entities continues primarily using UML model artifacts. In that it is
difficult for software to contain hardware, the use of TSA as the entry analysis probably makes more sense.

At the lower tier UML analyses where the physical and logical classifiers are classes and objects, the lines that flow between the classifiers on the corresponding sequence and communication diagrams coordinate with messages and influences applied to/from those classes in relation to external physical and logical classifiers (other classes generally at this level). This same effect is in operation whether the classifiers depicted are classes, components, or nodes. Each classifier must have identified for it one or more operations (services or functions) that it performs relative to an outside set of actors and one or more data entities it deals with internally to accomplish those operations. The data elements will flow into the classifier via the lines on the sequence and communications diagrams and data may be created or altered internally. Initially, the analyst may choose to first identify node, component, class responsibilities and subsequently as the analysis matures translate these responsibilities into operations and attributes.

In lower tier analyses, the assigned IPPT are responsible for identifying and defining needed interfaces below the level initially identified by the PIT. Where the two terminals of an interface touch only entities that are the responsibility of a single team, that team is clearly responsible for interface identification, definition, and integration. Where an IPPT is responsible for only one terminal of an interface, that team must cooperate with the team responsible for the other terminal to develop that interface. The integration agent in this case is the lowest common team. If there is only a single layer of teams under the PIT, the PIT is always the lowest common team. In general, it should be the team on the receiving end of an interface that first identifies the need for a new interface since interfaces should not be defined based on what is available but what is needed. If it is not obvious what team shall be responsible for the new interface, the PIT shall act as the integration agent until such time as the source is identified and then the lowest common team will take over that responsibility.

As IPPT identify lower tier entities during use case analysis, the PIT shall approve those additions and assemble them into the formal product entity structure. These entities may be physical or logical entities but eventually all of them must be identified as real product entities that will be developed. These final entities can be logical entities as in the case of computer software that will run on a particular hardware computer entity. Where it appears that lower tier entities will entail significant complexity, new IPPT may be formed by the PIT that will be subordinate to the appropriate existing IPPT. Those lower tier IPPT will take over the continuing analysis of use cases appropriate at that level and the immediately superior team will take on the role of the system engineer for the new team just as the PIT does for all top tier IPPT.

The use case analysis process employed by an IPPT or a collection of teams will necessarily be a collaborative process involving people from several different specialty disciplines. Each such team will have a leader whose responsibility it is to bring the analysis to a conclusion as scheduled. These teams will come into being, exist for a brief period of time, come to a conclusion, and pass from the scene with others replacing them. Once a use case analysis has been completed by a team, the work products will have been captured in modeling applications and integrated relative the existing work products. The modeling front will move down through the advancing product entity structure till the system is fully characterized.
Where personnel from other teams are required to accomplish work on another team's use case analysis, the owning team will cover the manhours of all personnel working the use case. Where all team members are physically collocated in the same facility, they may be depended upon to interact well through a traditional meeting in the common facility. Where at least some of the people required on a use case analysis effort are not collocated, it will be necessary to extend the meeting place geographically through the use of a product such as webex where people from several different geographical locations can cooperate in the development of a set of information.

Leaders of use case analysis teams are responsible for the prompt completion of team activity with good results but in so doing they will be well served to identify the most effective collaborative engineer on the team to lead collaborative team activity in the form of meetings especially where those meetings entail the use of distance integration aids like webex. The collaborative discussions in meetings need not necessarily be led by the team leader.

As the team completes its modeling and requirements work, the results should be reviewed by the parent team and, if approved, the team should be empowered to proceed with design at a level appropriate to the team responsibilities. The team must continue any responsibilities it may have for integration and optimization and leadership of subordinate teams that may still be involved in modeling work. This design work will entail some combination of hardware and software design development.

2.2.3 Dynamic Modeling Artifacts Explained

The use case diagram is considered a dynamic modeling artifact also but it is treated here as a transition medium between classifiers and the dynamic model set. The remaining dynamic models are implemented in four diagrams from which we may select any subset including all of them, any one of them, any pair, or any trio for a particular scenario analysis. It is not necessary to use them all for any particular analysis. Use those that make it possible to understand the problem space and properly characterize it in requirements. Some very large programs have done much of the analysis with only use case and sequence diagrams. The more complex the problem space and the more intimately the parts of the evolving system interact, the more different views of problem space that will be useful.

The first two kinds of diagrams covered, sequence and communications are both modeling the same relationships and collectively referred to as interaction diagrams. The sequence diagram emphasizes the time ordering of messages and the communication diagram emphasizes the organization of the objects that participate in the interaction. The second two are forms of state diagrams in the mind of many analysts.

2.2.3.1 Sequence Diagram

Some programs apply only the sequence diagram to explore the dynamic behavior of use cases and this may be adequate on relatively simple problems. In this tutorial we are assuming that the analyst employs the sequence diagram as the initial dynamic model, though an alternate approach would be to use the activity diagram for that purpose, but uses at least some of the
other three models to explore the use case more thoroughly. The sequence diagram example in Figure 23 illustrates the fundamentals showing two classifiers that comprise the classifier AX that has previously been analyzed. Here we conclude that AX must consist of AX1 and AX2 and that in order to accomplish the behavior defined for AX these lower tier classifiers must interact with an outside entity called an actor which will derive some kind of benefits from the relationship. The two subordinate classifiers will provide certain operations that are not clearly defined on this diagram. In the process of doing so, they will exchange messages in a certain order with time running down the page.

Each classifier including the actor has a lifeline in the form of a dashed line running down the page. A block is overlaid on this dashed line to indicate the active period(s) of classifier. Between these classifier lifelines we see messages being passed from one classifier to another. The names of these classifiers are formed of one or more words concatenated together without spaces and all but the first word capitalized. After the message name one can insert an argument list parenthetically.

The model permits the creation of classifiers and when they have performed their activity they can be killed. These features are not illustrated in Figure 23. The kinds of messages identified are: (1) a call invokes an operation on a classifier on the arrow end of the message, (2) a return message returns a value to the caller (dashed line used), (3) a send message sends a signal to a classifier, (4) a create message creates a classifier, and (5) a destroy message destroys a classifier. Message two and four could be an example of messages types 1 and 2. Message three is an example of message type 1 where the classifier is making the call upon itself.

After a classifier sends a signal to another classifier the sending classifier returns continues its own execution. The target classifier independently decides what to do about it. A common reaction would be to trigger a state machine causing the target classifier to execute actions and change state.
2.2.3.2 Communication Diagram

In some cases we are primarily interested in the time ordered sequence of messages between classifiers but in other situations we may be more interested in the organization of the classifiers and a communication diagram can offer better results even though the sequence and communication diagrams are semantically equivalent. Figure 24 illustrates a communication diagram reflecting the same situation as Figure 23. The classifiers are joined by lines corresponding to the relationships between them and messages are related to these line each in terms of message name, direction, and the relative timing of the message.

![Communication Diagram Example](image)

2.2.3.3 Activity Diagram

An activity diagram can be used to express the things that one or more classifiers must accomplish. It is weak in terms of absolute timing of accomplishing those things but strong in expressing the relative order of those things. As shown in Figure 25, activities are illustrated by rounded corner boxes and they are connected into a sequence by directed line segments. In addition to the activities, we also wish to show simultaneity through the use of forking and joining structures and alternative paths using branch and merge structures. The guard expressions give information about the conditions that correspond to movement through one branch or another.

This is the functional flow diagram of UML though many analysts prefer to think of the blocks as states rather than functions. The author prefers not to consider them states because it is in conflict with the notion that an entity must be in only one state at a time.

The diagram can be built with swim lanes, or not, that relate to classifiers, the same classifiers identified on sequence diagrams using the lifelines. It is through these two devices that we can allocate software functionality to classifiers. As we do so we determine the next lower tier product entity structure and should offer up newly identified entities to the PIT for inclusion in the product entity structure.
2.2.3.4 State Diagram

An interaction diagram (sequence or communication) models the relationships between a collection of classifiers while a state machine models the behavior of a single classifier. The classifier in this case can be the whole system or a classifier at any level of abstraction. The state machine models the possible condition that a classifier can exist in and the transition of that classifier from one condition or state to another based on a stimulus that might be a signal from another classifier, the passage of time, receipt of a call message invoking an operation, or a change in some condition.

A state is drawn on a state diagram, as shown in Figure 26 showing a state machine for temperature control, as a rounded corner box with the name of the state written inside. Generally the diagram must have an entry and final state symbols though it is possible that once entered the state machine may continue forever. In some cases the intent may be for the machine to continue forever, as in a traffic light system, the reality is that such a system may have to be shut down for maintenance and does need a final state. Transitions between states triggered by events in the life of the classifier being modeled are shown diagrammatically as directed line segments between a pair of state and named in a way to convey how that transition is triggered. It is understood that the machine must be in only one state at a time and that only a single transition is possible at one time. Transitions are generally thought of as taking place instantaneously.
While UML does not prescribe it, a pair of dictionaries can be helpful in clearly stating the intended operation of a state diagram. One dictionary lists the states and defines them with precision while the other lists and defines the transitions. You will note that the transitions in Figure 26 have not been named uniquely but should be in the general case so that each can one can differentiate between them in such a listing.

2.2.4 Static Entity Analysis

In early object oriented analysis (OOA) as prescribed by Grady Booch, Peter Coad teamed with Edward Yourdon, James Rumbaugh, et. al., and many other practitioners, the proper way to enter problem space was using the static view with objects. Then they encouraged the analysis of the objects from a dynamic perspective with data flow diagrams for functionality and state diagrams for behavior. This approach is possible with UML using the foursome of dynamic modeling artifacts discussed in the prior section and it can be effective when developing a largely preceded system that can be observed in the real World like a new payroll system. The author believes that largely unprecedented problems are best attacked using Sullivan's encouragement of form follows function leading with the dynamics views and identifying the static entities that populate the product entity structure from a software perspective.

UML identifies three levels of static entities but they are all product entities and while drawn on modeling diagrams using different images, they are essentially the same at different levels of abstraction. All are simply illustrated on the system product entity diagram illustrated in Figure 14 as blocks. Figure 27 illustrates the three static entities collectively referred to as classifiers in this tutorial.
In this tutorial the case is made for first identifying the nodes which are entities that will be associated with run time software. They are higher-tier entities. Like classes and components, they have associated attributes and operations. They interface with each other and possess lower tier interfaces between components and classes that comprise them. The nodes are identified in the dynamic analysis of the top-level software entity by identifying sequence diagram lifelines and activity diagram swim lanes. We then analyze these nodes from a dynamic perspective and identify components in the same fashion.

The alternative approach first identifies classes corresponding to observable entities in the problem space which are then dynamically analyzed leading to an understanding about how best to package these entities based on collecting the classes with the most intense interface relationships together as components. Just as in the use of interface analysis in TSA to validate the product entity decisions in functional analysis by observing possible unintended interface intensities, we can in UML re-consider the particular swim lanes and lifelines we selected in the dynamic analysis.

In this section, the intent is to explain what the UML static entities are and how they are used on diagrams. We will use classes in order to do so with the understanding that nodes and components are but higher tier classes. First we will describe a general class then explore structural relationships between these classes and finally we will cover the messages that are passed between them.

2.2.4.1 The Class

A class is illustrated as a box as shown in Figure 27c. The name of the class is placed in the top portion of the box, attributes are listed in the middle portion of the box, and operations are listed in the lower portion of the box. A forth box can be included below the operations in which we inscribe class responsibilities in free-form text. A responsibility is a contract or an obligation of a class. You may find it useful to begin the analysis of classes this way translating these into attributes and operations that best fulfill the class’s responsibility as the model is refined.
2.2.4.2 Class Relationships

Figure 28 illustrates the structural relationships recognized between classes. A class is said to be dependent on another if it depends on that other class for information or services. A class can be linked hierarchically to another through a generalization. Class associations can have the three adornments illustrated in Figure 29.

NameOne is the base class. NameTwo and NameThree are leaf classes in a generalization.

An association is a structural relationship

- **Association Name**
  - NameOne
  - NameTwo

- **Association Role**
  - The face that the class at the far end of an association presents to the class at near end of the association. Role names called end names.

- **Association Multiplicity**
  - Tells how many objects may be connected across an association instance. Given by a range of numbers.

- **Association Aggregation**
  - Expresses a whole-part relationship between to associated classes.
2.2.4.3 Messages

Classes can also be related through the five kinds of messages discussed in Paragraph 2.2.3.1. Such a message can convey to a class a variable argument (value) that is needed in a class operation or it can convey a signal that causes the class state machine to transition to a new state for example. The messages that must be passed between classes are understood in the context of the sequence diagram under the assumption that the dynamic analysis is accomplished prior to the static analysis.

2.2.5 Related Analyses

2.2.5.1 Specialty Engineering

The specialty engineering matrix discussed in paragraph 2.1.9 can be used in software as well as hardware to identify all product entities for which specialty engineering requirements must be developed. This includes, for example, safety, reliability, and security. The software interface requirements fall out of the sequence and communication diagram analyses and flow into the specification template offered in Figure 27.

2.2.5.2 Environmental Requirements

Software environmental requirements are somewhat different from the hardware and system environmental requirements that tend to be dominated by the natural environmental factors. The software being an intellectual entity rather than a physical one, is shielded from the natural environmental stresses. True, the software operating within a machine that can suffer adverse environmental stresses can as a result fail, but this is a secondary effect. Reasonable software environmental relationships include any language restrictions and the computer architecture upon which it must run, for example.

2.2.6 Specification Structure

The specification outline offered in Figure 5 can also be applied to software entities where the capabilities are linked to either the terminators, use cases, extended use cases, or scenarios and the subordinate requirements listed under each capability are drawn from the corresponding dynamic diagramming (activity, sequence, communication, and state diagramming) work. Thus the requirements can be clearly shown to trace to modeling artifacts just as the performance requirements in hardware can be shown to flow so clearly from functions. Figure 30 offers an outline for a software requirements specification (SRS) using an edited template from EIA J STD 016 to integrate the supporting modeling work into the specification. Note the similarity to the outline in Figure 5 for a system or hardware item performance specification.
<table>
<thead>
<tr>
<th>PARA</th>
<th>TITLE</th>
<th>MODEL</th>
<th>DEPT</th>
<th>APP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>SCOPE</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>2.</td>
<td>APPLICABLE DOCUMENTS</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.</td>
<td>REQUIREMENTS</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1</td>
<td>Required states and modes</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.1</td>
<td>Classifier context diagram</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.2</td>
<td>Use case analysis</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.2.h</td>
<td>Terminator h</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.2.h.i</td>
<td>Terminator h, use case i</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.2.h.i.j</td>
<td>Terminator h, use case i, extended</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.2.h.i.j.k</td>
<td>Terminator h, use case i, extended</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.3</td>
<td>Dynamic Analysis</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.3.h</td>
<td>Terminator h dynamic analysis</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.3.h.i</td>
<td>Use case hi dynamic analysis</td>
<td>UML</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.3.h.i.j</td>
<td>Extended use case hij dynamic analysis</td>
<td>UML</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.3.h.i.j.k</td>
<td>Scenario hij dynamic analysis</td>
<td>UML</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.3.h.i.j.k.1</td>
<td>Sequence diagram hijk1 analysis</td>
<td>UML</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.3.h.i.j.k.2</td>
<td>Communication diagram hijk2 analysis</td>
<td>UML</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.3.h.i.j.k.3</td>
<td>Activity diagram hijk3 analysis</td>
<td>UML</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.3.h.i.j.k.4</td>
<td>State diagram hijk4 analysis</td>
<td>UML</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.4</td>
<td>Lower tier classifier identification</td>
<td>UML</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.2</td>
<td>Entity capability requirements</td>
<td>UML</td>
<td>2100</td>
<td>A</td>
</tr>
<tr>
<td>3.2.m</td>
<td>Capability m</td>
<td>UML</td>
<td>2100</td>
<td>A</td>
</tr>
<tr>
<td>3.2.m.n</td>
<td>Capability m, requirement n</td>
<td>UML</td>
<td>2100</td>
<td>A</td>
</tr>
<tr>
<td>3.3</td>
<td>Interface requirements</td>
<td>UML</td>
<td>2100</td>
<td>D</td>
</tr>
<tr>
<td>3.3.1</td>
<td>External interface requirements</td>
<td>UML</td>
<td>2100</td>
<td>D</td>
</tr>
<tr>
<td>3.3.1.m</td>
<td>External interface m</td>
<td>UML</td>
<td>2100</td>
<td>D</td>
</tr>
<tr>
<td>3.3.1.m.n</td>
<td>External interface m, requirement n</td>
<td>UML</td>
<td>2100</td>
<td>D</td>
</tr>
<tr>
<td>3.3.2</td>
<td>Internal interface requirements</td>
<td>UML</td>
<td>2100</td>
<td>D</td>
</tr>
<tr>
<td>3.3.2.m</td>
<td>Internal interface m</td>
<td>UML</td>
<td>2100</td>
<td>D</td>
</tr>
<tr>
<td>3.3.2.m.n</td>
<td>Internal Interface m, requirement n</td>
<td>UML</td>
<td>2100</td>
<td>D</td>
</tr>
<tr>
<td>3.4</td>
<td>Specialty engineering requirements</td>
<td>DID</td>
<td>2100</td>
<td>E</td>
</tr>
<tr>
<td>3.4.m</td>
<td>Specialty Engineering Discipline m</td>
<td>Specialty Scoping</td>
<td>E</td>
<td></td>
</tr>
<tr>
<td>3.4.m.n</td>
<td>Specialty Engineering Discipline m,</td>
<td>Specialty Scoping</td>
<td>E</td>
<td></td>
</tr>
<tr>
<td>3.5</td>
<td>Environmental conditions</td>
<td>3-Layered Env Model</td>
<td>2100</td>
<td>B</td>
</tr>
<tr>
<td>3.6</td>
<td>Precedence and criticality of requirements</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>4.</td>
<td>VERIFICATION</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>5.</td>
<td>PACKAGING</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>6.</td>
<td>NOTES</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>6.1</td>
<td>Requirements traceability</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>6.1.1</td>
<td>Inter-specification traceability</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>6.1.2</td>
<td>Verification traceability</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>6.1.3</td>
<td>Modeling traceability</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>6.1.4</td>
<td>Section 2 traceability</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>6.1.5</td>
<td>Programmatic traceability</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>6.2</td>
<td>Glossary</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>6.3</td>
<td>Specification maturity tracking</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
</tbody>
</table>

Figure 30  Software Requirements Specification (SRS) Template
Refer to Exhibit B of this student manual for a JOG System Engineering data item description (DID) that shows how the subparagraphs of paragraph 3.1 relate to the subparagraphs of paragraph 3.2. Paragraph 3.1 essentially provides an opportunity to capture the complete UML analysis for the classifier covered in the specification. This can be done by actually including the diagrams discussed in prior paragraphs in this text or by referencing them in a computer application within which the modeling is done and derived requirements captured or the diagrams can be completed manually (or with a computer graphics application like Microsoft Visio or Powerpoint) and contained in the appendices of the system definition document (SDD).

The reader will note that in Figure 19 there is a second plane labeled UML for the case where the program chooses to capture the UML analysis work products in the SDD. On a program that is dealing only with a UML analysis for a software product the Appendix structure shown in Table 1 column A could be used. If the program must deal with both TSA and UML, the software appendices could be simply added to those required for TSA and lettered G through N with the TSA Appendix G (the RAS that should also contain the software requirements derived from UML artifacts using the MID pattern illustrated in Figure 22) becoming a common RAS in Appendix O. The latter pattern is captured in column B. In this combined case, the classifier diagramming can remain in Appendix N but Appendix C should capture the aggregate product entity structure including hardware and software entities.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>MODEL ARTIFACT</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>G</td>
<td>Context Diagrams</td>
</tr>
<tr>
<td>B</td>
<td>H</td>
<td>Use Case Diagrams</td>
</tr>
<tr>
<td>C</td>
<td>I</td>
<td>Scenarios Text</td>
</tr>
<tr>
<td>D</td>
<td>J</td>
<td>Sequence Diagrams</td>
</tr>
<tr>
<td>E</td>
<td>K</td>
<td>Communication Diagrams</td>
</tr>
<tr>
<td>F</td>
<td>L</td>
<td>Activity Diagrams</td>
</tr>
<tr>
<td>G</td>
<td>M</td>
<td>State Diagrams</td>
</tr>
<tr>
<td>H</td>
<td>N</td>
<td>Classifier (Object, Class, Component, and Deployment) Diagrams</td>
</tr>
<tr>
<td>I</td>
<td>O</td>
<td>RAS</td>
</tr>
</tbody>
</table>

The reader should note that in software using UML it is possible to relate the modeling machinery very clearly with the classifiers about which specifications are to be written. It is not quite so easy in hardware using TSA because all of the performance requirements derived from a particular function may be allocated to the same entity. It might be possible to force fit the functions into a closer alignment with the product entity structure by restarting the functional analysis for each entity identified but the author does not think this would be particularly helpful.

2.2.7 Software Requirements Close-Out

When a set of classes that collectively form a component have been thoroughly characterized including the sets of requirements associated with those classes and dynamic relationships and the related requirements have been captured, reviewed, and approved, the responsible team can
start to develop the computer code that will implement the component. This process can begin across the lower tier of the components and continue to next assemble into computer code for the nodes. This process may take place in a single string or simultaneously following the same pattern in multiple paths where the software is distributed to some extent. There is, of course, a continuing need for integration and optimization across the development being accomplished by different teams that focus primarily on interfaces just as in hardware development.

The PIT and program manager will reach a conclusion for the program with possible local differences whether to pursue software design and coding in a top-down or bottom-up fashion. In either case, any code developed must be tested at each level of build. This may require the development of special stub (top-down) and/or drivers (bottom-up) to permit testing of completed portions before higher or lower tier software has been completed.

The classifiers identified in the UML analysis should be entered into the product entity structure by the PIT so that a consolidated view of the overall system matures. Figure 31 shows the product entity structure evolving based on feed from all modeling activities being applied on the program under the watchful integrating eye of the PIT.

![Figure 31 Evolving Product Entity Structure](image)

The UML analysis process encouraged does involve a partitioning of the whole problem space into parts hierarchically arranged and assigned to teams. These teams, in hardware and software, can generally be depended upon to integrate and optimize at the level for which they are responsible rather than the next higher level. Therefore the PIT (and superior IPPT) must integrate and optimize across the best efforts of the teams. The PIT should also consider special integration studies across the grain of the system based on overarching functions such as shutdown and activation as well as others that may be suggested through system level states and modes analysis. These efforts should be led by the PIT and participated in by people from a wide range of teams.

2.3 Opening the Analysis With DoDAF

DoDAF was developed to support modeling of the complex information systems that DoD has been assembling in recent years to interconnect their many sensor and weapons systems to form
an effective military capability often referred to as sensor-to-shooter connectivity. It does not provide a complete modeling set primarily because of the absence of a model artifact for the physical product entities. The product entity view can be provided by augmenting the DoDAF model set with the TSA product entity diagram, however, from the aggregate model set this description assembles. Physical products identified in the DoDAF process can be inserted into the TSA product entity diagram just as they can be using any of the model sets and as suggested in Figure 31.

The DoDAF problem space entry is similar to the UML entry and in fact can use some UML artifacts to do so. DoD does not prescribe any particular set of modeling artifacts for the several views. In this paper we will use a combination of UML, IDEF, and TSA artifacts. DoDAF employs four modeling views: (1) two all view products that offer a textual overview of the system (as in what was once called a mission need statement) and an overall glossary explaining all terms used; (2) two technical view products that define standards that must be respected on the program and the evolution of those standards over time; (3) nine operational view products that capture how the user views their needs; and (4) thirteen system view products that offer the engineering perspective on the needed system. Figure 32 illustrates a recommended DoDAF development sequence. The simulation work could better be shown overarching both the analysis and design work interacting with and serving both. Note that the two all and two technical views would be a good addition to any modeling apparatus.

![Figure 32 DoDAF Development Sequence](image)

The technical and all views work products should be developed initially by the user and acquisition agents and evolved by the contractors selected to develop the system. The user should have developed the operational views in the process of evolving a set of documents used to refine their need. These documents include a Joint Capabilities Document (JCD that takes the
place of the old need statement cut very broadly. In the study process prescribed by DoD for a new system, the capability identified in the JCD may be cut up into n systems each of which should have developed for it an Initial Capabilities Document (ICD) that is the equivalent of the old mission needs statement for a given system to be procured. The ICD is then matured into a Capabilities Development Document (CDD) and eventually a Capabilities Production Document (CPD). In between these last two a contract would commonly be let resulting in the development of a system specification attached to the contract. DoDAF can be used to gain a systematic insight into the content needed in all of these documents.

2.4 Integrated Modeling

A program must make a decision about the modeling techniques it will apply as it builds the proposal. This may include some mix of TSA, UML, SysML, DoDAF, and IDEF-0. A program that is going to be primarily computer software or networked assets could enter the program with UML or DoDAF. If the product is a database, it could enter with IDEF-0 or UML. But, generally, modeling entry should involve the use of TSA or SysML at the system level because software, an intellectual entity, must run on hardware entities that provide the product with real substance. As noted earlier, at the time this was written SysML was not yet fully ready for use so TSA would be the author's preference now. But, an enterprise should continue to follow the development of SysML and work toward replacing TSA with SysML. In any case, there is a need to recognize that for some period of time there will be a need for a model that works well for systems and hardware and another model that works well for SW. For now, also, there is not one great computer application within which one can model HW and SW requirements work and permit easy cross model traceability and provide specification publication capability so it will be necessary to use two or three applications to cover the needed tool set.

Work can be accomplished well for systems and HW as covered under paragraph 2.1 and for SW using the approach covered under paragraph 2.2. When applying both, problems will tend to arise when transitions have to be made between these two approaches. It is not possible for SW to include HW but the opposite case is perfectly normal. So, the transitions will only be a problem as the analysis shifts from HW to SW moving from the use of TSA or SysML to UML. There are two concerns at this point: one in the models applied and the other in the computer applications employed.

The transition point will occur when the highest tier software entities are identified. There may, of course, be several of these transitions distributed about the expanding product entity structure. The program has the option of pooling all of the software into an integrated entity or permitting it to be distributed within multiple processors that may still all be under the responsibility of one team or distributed among teams with both hardware and software responsibilities. If we can solve one of these hardware-software handoffs we will have solved the general problem of requirements traceability across these gaps.

It should be clear that requirements traceability to models is assured in the approach covered in this tutorial because all of the requirements are to be derived from a model. Vertical or hierarchical requirements traceability is very simple in specialty engineering areas in hardware, software, and across the gap as described in this tutorial. The environmental requirements are
vitaly different between hardware and software and one can make a case that lower tier software environmental requirements should not have to respect traceability across the gap to higher tier hardware or system environmental requirements that are largely environmentally related. Precisely the same method of identifying hardware interface requirements can be used to identify software interfaces as well as hardware-software interfaces because we identify them between entities that appear in the joint product entity structure. So, if interface requirements traceability involves lower tier interface expansion requirements to higher tier interface requirements, traceability is assured. This leaves only the performance requirements a remaining problem from a vertical or hierarchical traceability perspective.

Given that the system entry analysis was accomplished in TSA using some form of functional analysis and the lower tier software analysis is going to be done using UML, there is a temptation to employ activity diagrams in UML to analyze software entities from a dynamic perspective because it is very similar to functional flow diagramming and might give us some interesting opportunities to link up hierarchical traceability. However, for a given software entity there may have been 10 performance requirements derived from 8 functions allocated to the software entity in question. There is no clear way to link up the activity analysis and requirements derived from it with the several functional analysis strings and the performance requirements derived from them that can easily be automated.

So, let us pursue another tack in an attempt to coordinate the traceability relative to the sequence oriented dynamic analysis approach described previously. If requirement $R_{@1}$ is one of a set of requirements $R_{@1}$ through $R_{@10}$ where $R_{@1}$ is derived from function $F_{#1}$ of a set of functions $F_{#1}$ through $F_{#8}$ and requirement $R_{@1}$ is allocated to product entity AX2 and it is decided that AX is going to be developed as a software entity, then one of the scenarios to be analyzed will be UXhijk. Assume that we accomplish the dynamic analysis using sequence diagram UXhijk1 from which we derive requirement $R_{@1}$. What we are looking for is a way to establish hierarchical traceability between requirement $R_{@1}$ and some requirement in the set $R_{@1}$ through $R_{@10}$. The X, %, and # characters are being used to designate base 60 strings in this discussion. We know that requirement $R_{@1}$ must be traceable to one of the 10 performance requirements allocated to classifier AX2 and we can look at that list of requirements and select the one most closely related.

To make this selection more organized, we can form an x by y matrix, in this case a 10 by 12 matrix, and pair-wise compare the sets R@ and R%. In Figure 33 you can see this whole process taking place. The 10 functionally derived requirements are captured in the RAS mapped to the set of functions $F_{#1}$ through $F_{#8}$ and allocated to product entity A&. Based on these requirements we build a context diagram for entity A& and analyze A& from the perspective of each of the three terminators shown. As an example Use Case U&3 is extended to three use cases and we build three scenarios one of which, U&3111 is analyzed from a dynamic perspective with some combination of sequence, communication, activity, and state diagrams. Requirements $R_{@1}$ through $R_{@12}$ are derived from these analyses and captured in the RAS (possibly linked to the RAS database from a UML modeling application).

There are 10 requirements ($R_{@1}$ through $R_{@10}$) to which the requirements in the set $R_{@1}$ through $R_{@12}$ will have to hierarchically trace. We can build a 10 x 12 matrix and pair-wise analyze the
relationships between the two sets of requirements, perhaps concluding that one of the matches is $R@_1$ traces to $R%_1$. All of the matches are marked in the requirements management database table for traceability relationships. There are no known databases that provide the traceability evaluation matrix so it may have to be accomplished as a pencil and paper aid. However we should be able to set the requirements management database filter for the two sets of requirements of interest aiding in the identification of the sets of interest for a particular case. In this example, there might be ten or more sets of requirements like $R@$ derived from ten or more dynamic analyses. In each case an $x$ by $y$ matrix would be needed to pair-wise analyze the traceability relationships. In any case, it should be clear that we can have good requirements to modeling traceability and even good hierarchical traceability across the gap between performance requirements.

![Figure 33 Requirements Traceability Across the Gap](image)

In both TSA and UML we have discussed a decomposition process that partitions the problem space into parts in which the analysis is accomplished. Whenever we partition any whole we have an obligation to integrate and optimize across the boundary conditions thus created. The PIT must accomplish this integration work relative to the top level IPPT and each IPPT with lower tier teams must accomplish this work relative to it's own immediately subordinate teams. Much of this integration work will take place at the interfaces ensuring that requirements on one end of an interface are compatible with those for the other terminal. Each team with subordinate teams, however, should also integrate across its immediately subordinate teams relative to the requirements derived at the subordinate team level relative to those at the parent team level. Part of this work can be accomplished by simply establishing the traceability between the requirements at the two levels. Another approach of value is to accomplish higher tier function effects across the lower tier team responsibilities. For example, one can inquire collaboratively into lower tier performance of higher tier functions like turning the system or entity on or off,
moving from one major mode to another, accomplishing some kind of transfer function, or physical separation or joining of two entities.

Another kind of traceability can also be used to stimulate integrating results. This was pointed out to the author by an engineer at Puget Sound Naval Base in Bremerton, WA. Given a requirement at level m, we can inquire if the intent of the requirement was fully implemented in the requirements for the n entities at level m+1 (downward). This kind of traceability inspection must await the development of the subordinate specifications, of course, as does all hierarchical traceability.

At one time, in the 1950s when software was a very young discipline, it happened that hardware and software analysis, to the extent that it was done, was accomplished using exactly the same model, flow charting as shown in Figure 34. Over time, probably encouraged by the ease with which flow charts could be outputted onto line printers using ASCII symbols, computer software people got into the habit of building flow charts in the vertical rather than the horizontal axis still used by system engineering in their functional flow diagrams. The activity diagrams of UML still reflect the vertical orientation but it is really of little significance which orientation is used. The absolutely fascinating approaching reality is that system and software people will rejoin the same house in the near future. As UML and SysML become more fully integrated as suggested in Figure 34, we will achieve a tremendous milestone of universal unified modeling capability.

As we pass through this door into a world of integrated modeling and supporting computer applications, we will find it a more reasonably affordable task to integrate across the hardware-software boundary than has been the case for many years. But then as now, integration takes place in the minds of the system engineers working on the program. These engineers must be
every vigilant for inconsistencies between information sets that signal that two different domains are not working from a common understanding of the problem and solution spaces.

3 Requirements Management

3.1 Summary of Team Activity During Requirements Work

During initial product development work, the PIT will model the problem space using a predefined set of modeling methods selected from the list of enterprise-approved modeling methods and apply those methods to identify top-level system entities and interface relationships and their requirements. This level of system definition shall be completed before program level IPPT are initiated on the program and staffed. These top-level product entities are the basis for assignment of these teams. When a team is established, the leader shall be presented with the specification for the top-level entity for which the team shall be responsible, a design concept (in particular if it is HW or SW), a clear definition of all external interfaces, and the corresponding components of the WBS, SOW, IMP, and IMS. The team will be charged with the development of that item and all subordinate entities and interface relationships. In the process of so doing, the team may conclude that lower tier teams are required and must request that action of the PIT.

The work products from the IPPT will be loaded into the computer applications used on the program by PIT and checked for consistency. The PIT shall perform integration and optimization work on modeling work contributed by the IPPT fitting the work products into a coherent system analysis of the problem space. As part of this work where the product is to be implemented in SW, the responsible team will seek to identify all needed use case analyses and assign them for completion by people on the team if possible. Where this work must involve people from other
teams, the responsible team must request help from the PIT and a cross team analysis effort will be established. The responsible lead team must ensure that each analysis is complete with all needed supporting modeling work.

The PIT shall maintain the product entity structure, the interface relationships, and all requirements modeling and management assets. The requirements shall be retained in a relational database from which specifications may be printed to paper or computer screen and within which traceability may be maintained. This database shall be linked to one or more modeling applications used on the program. The modeling applications shall be used for the purpose of identifying the system entities, interfaces, and appropriate requirements in each case. All content of these applications shall be under the control of the PIT until such time that it is formally approved at which time it shall fall under configuration management control and shall not be changed without a formal approval as well. Responsibility for data entry can be distributed to IPPT or retained by PIT. Entry may be aided by special Microsoft Office applications making it unnecessary for personnel to develop and maintain computer application skills.

The PIT will seek to establish IPPT overlaid upon the product entity structure so as to minimize the interface relationships between the entities for which the teams are responsible. The purpose of this arrangement is to minimize the need for cross team coordination and staffing for use cases analyses.

The preferred lower tier HW development approach entails a continued application of TSA using the same modeling database application applied at the system level. The preferred SW product development approach entails PIT and IPPT application of unified modeling language (UML). In the near term traditional structured analysis (TSA) will be applied in combination with UML maturing to a combination of UML and SysML as the latter matures into a formally released model by the Object Modeling Group. TSA or SysML are to be used initially to identify system and top level product entities that will more often be hardware end items. As the analysis proceeds downward and identifies a need for computer software, the analysis should switch to UML.

3.2 Requirements Tools Base

Figure 36 illustrates the preferred tools environment for programs. A requirements management database is used to capture all of the requirements that will be published in specifications and those specifications may be published from this database. In Figure 36 this is referred to as a big dumb database with no slur intended for the makers of tools that do not include integrated modeling capabilities. This is a relatively simple relational database application that can contain text information in a tabular structure. Each table row corresponds to a unique requirement with data captured in table columns for the several fields needed. Additional relational tables may be required for vertical traceability, verification traceability, and lateral (to models from which the requirements are derived) traceability. The program may use available modeling applications for UML (such as Rational Rose) and TSA (such as CORE) and arrange for traceability between these applications and the management application in an application like DOORS.
Many enterprises find it difficult for engineers to maintain currency with a set of requirements database applications as well as other applications more directly related to their work. All or most of these engineers will, as a function of accomplishing their normal work, maintain proficiency with the three fundamental Microsoft Office applications (Word, Powerpoint, and Excel). Loader applications crafted from Microsoft Office applications may be used to permit all engineers to enter data into the requirements database suite without a need for the engineers to become intimately familiar with these applications.

The PIT must exercise integration and optimization control over the requirements application suite and will require some members who really understand the applications, how they work, and how their content is inter-related. The suit must be set up to permit passing control of approved content to configuration management while retaining control of all in-work data.

3.3 Recommended Specification Responsibility Pattern

In the author's view, a program should staff a program integration team (PIT) that should begin the requirements analysis process at the system level and develop the top level diagrams in the SDD. This work should continue as necessary to develop the content of the system specification and the specifications corresponding to the top-level teams. The structured analysis for each of these teams should be taken over by the corresponding IPPTs in each case until they have completed the content of the specifications that define the problem for any subordinate teams. If no subordinate teams have been identified then they would have to complete the analysis needed to develop all of the specifications subordinate to their top-level specification. This same pattern
carries down to the lowest level. Each team should act as the system agent for all of its lower tier teams and principal engineers. This starts at the PIT for the system and works its way down through the lower tier teams. The Program Manager and Chief Engineer/PIT Manager should review and approve the system specification and all top-level specifications. PIT should establish rules for review and approval of lower tier specifications created by the teams.

With different parties doing the structured analysis, it is necessary to apply process integration and the PIT should do that accepting data into the several appendices of the SDD, numbering the figures, and cross-checking the data submitted. At least one team will be involved in software development and if traditional structured analysis has been applied for the system level, then that or those teams responsible for software will want to switch to some form of software modeling such as UML. Regardless of the modeling methods applied, all of the requirements modeling artifacts should be captured in the SDD either in the paper appendices noted earlier or referenced in the database systems used. The integrated RAS should be implemented in the big dumb database of Figure 36 as simply the basic relational database table used to capture requirements.

3.4 Requirements Risk Management

The principal risks that appear during the requirements development work involve a sensed inability to satisfy the requirements. The risk may be motivated by the conclusion that insufficient financial, or schedule resources have been made available. The concern may be that the requirement simply cannot be satisfied with available technology. Finally, the concern may be motivated by the conclusion that the value is simply too hard to achieve with available skill and knowledge. It is not uncommon to partition all into the categories of cost, schedule, technology, and performance as a result.

3.4.1 Requirements Validation

EIA 632 identifies an activity called requirements validation where we make an effort to determine to what extent we can satisfy particular requirements. The simplest way of reaching this conclusion is to simply ask the person(s) responsible for accomplishing the related design work if they can satisfy the requirements. If there is a lack of confidence then we need to proceed further in our efforts to identify potential performance risks. As requirements are identified and written we should validate them at that time. In most cases, the conclusion will be that there is no problem. Should we conclude that either there is a problem or we are not certain that can satisfy the requirement the first alternative investigated should be to ask if the requirement can be changed making it more certain that it can be satisfied. If that is not possible, then we should choose a means to mitigate the risk through an appropriate analysis, development evaluation test, simulation, or demonstration. If we believe that the requirement is very important in the development effort and that it will require some time to reach a final conclusion, we may select the requirement for more intense management as a technical performance measurement (TPM).

Parameters are managed through TPM by placing them on a list and assigning each TPM to a specific engineer who is charged with closing the gap between the required value and the currently demonstrated capability. The parameter principal engineer must maintain two charts: (1) a parameter chart that tracks these two values over time annotated with notes citing important
events coinciding with significant changes in the relationship between the values and (2) an action plan stating what is going to be done, when, and to what anticipated effect.

3.4.2 Margins and Budgets

Every program manager will experience difficult problems each requiring a tough decision. These problems can be made less severe by ensuring that the program manager has the resources available. This outcome is encouraged where the values for the most difficult requirements are combined with margins such that there is an opportunity to award engineers with a very difficult design problem some slack. These margins come in three varieties: cost, schedule, and performance. Cost margin is commonly applied as a management reserve such that the program manager can award a team more cost to solve a problem. Similarly, if a team has a design problem that can be solved through award of schedule slack time, the design may be possible. The third kind of slack is requirements margins. In all of these cases, the margin is derived by invariably making the job more difficult. Cost margin is often made available by skimming the task estimates of 10-15%. Schedule slack is obtained by subtracting available time for tasks on the critical path. Risky requirements are made more difficult to achieve. For example a weight margin may be realized by subtracting 5% from all weight figures. So the engineers will be challenged to accomplish their design with a weight value of required value - 5%. This is more difficult, clearly. The good news is that engineers will most often make these more demanding requirements preserving the margin values for the most difficult problems. When a very difficult weight problem appears, the manager can allocate some available margin. The margins invariably will be consumed before the design process is complete but there are ways of using available margins from requirements not of the same kind. For example, if an engineer has difficulty reaching his/her reliability figure after all of the reliability margin is gone, the manager can award some cost margin to permit the use of better parts or some mass margin permitting a heat sink that will reduce junction temperatures.

Requirements budgeting also has a risk reduction effect because it partitions available requirement values to the several designs at any one level of indenture. For example, given that it has been decided that 1500 watts of electrical power will be available from a source and there are 10 loads to be attached to this source, an engineer must partition this available power in a rational way between these several loads and integrate the results.

3.4.3 Risk Tracking

Risk is often measured using a dual axis criteria dealing with the probability that the concern will be realized and the degree of difficulty it will present if it does come to pass. This makes it a little difficult to track a single risk parameter over time and the way many people apply the dual axis system makes it difficult to accumulate a program metric that can be tracked over time. A variation on the safety hazard index described in MIL-STD-882 offers a way to measure risk with a single parameter that responds properly to characterize instantaneous values and a historical record for the program.

Figure 37 shows the risk matrix. Tables 2 and 3 corresponding provide the dictionaries explaining the values entered on the matrix axes. The intersections contain an index number that is simply the product of the axis numbers.
Table 2  Risk Probability of Occurrence Criteria

<table>
<thead>
<tr>
<th>CAT TITLE</th>
<th>P(O)</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 Nearly Certain</td>
<td>0.95-1.00</td>
<td>Will occur at least once during program</td>
</tr>
<tr>
<td>4 Probable</td>
<td>0.75-0.95</td>
<td>Will probably occur once during program</td>
</tr>
<tr>
<td>3 Possible</td>
<td>0.50-0.75</td>
<td>May occur during program</td>
</tr>
<tr>
<td>2 Unlikely</td>
<td>0.25-0.49</td>
<td>Will probably not occur during program</td>
</tr>
<tr>
<td>1 Nearly Impossible</td>
<td>0.00-0.24</td>
<td>Will not occur during program</td>
</tr>
</tbody>
</table>

Table 3  Risk Effects Criteria

<table>
<thead>
<tr>
<th>CAT TITLE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 Catastrophic</td>
<td>Program in jeopardy of termination</td>
</tr>
<tr>
<td>4 Serious</td>
<td>Serious damage to program</td>
</tr>
<tr>
<td>3 Moderate</td>
<td>Problems cause program focus difficulties</td>
</tr>
<tr>
<td>2 Minor</td>
<td>Problems that can be easily be overcome</td>
</tr>
<tr>
<td>1 Null</td>
<td>No problem</td>
</tr>
</tbody>
</table>
If you were to compare this information with the MIL-STD-882 safety hazard matrix, you would find that the safety hazard matrix offered in the military standard uses letters for one of the two axes and that the highest hazards (risks) have the lowest indices. Our matrix in Figure 37 uses numbers on both axes and the index values are higher for more serious risks. Therefore, it is possible to apply the index values in a mathematical sense as a program metric. Given the 6 program risks listed in the program risk list shown in Table 4 with the indicated risk index values, the instantaneous program risk index is 97.

Table 4  Program Risk List

<table>
<thead>
<tr>
<th>RISK NBR</th>
<th>RISK TITLE</th>
<th>PROB</th>
<th>EFF</th>
<th>INDEX</th>
<th>TM</th>
<th>PRINCIPAL</th>
<th>SUSP</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Life Cycle Cost</td>
<td>4</td>
<td>4</td>
<td>16</td>
<td>1</td>
<td>Burns</td>
<td>02-10-05</td>
</tr>
<tr>
<td>5</td>
<td>Payload Capacity</td>
<td>4</td>
<td>5</td>
<td>20</td>
<td>1</td>
<td>Adams</td>
<td>03-08-05</td>
</tr>
<tr>
<td>7</td>
<td>Stoddard Supplier Risk</td>
<td>5</td>
<td>4</td>
<td>20</td>
<td>3</td>
<td>Thornton</td>
<td>04-20-05</td>
</tr>
<tr>
<td>12</td>
<td>Program Funding</td>
<td>4</td>
<td>4</td>
<td>16</td>
<td>0</td>
<td>Connolly</td>
<td>03-10-05</td>
</tr>
<tr>
<td>15</td>
<td>Computer Software Schedule</td>
<td>5</td>
<td>5</td>
<td>25</td>
<td>4</td>
<td>Sampson</td>
<td>05-23-05</td>
</tr>
</tbody>
</table>

Thus, we have arrived at a program risk index or metric. If we maintain a chart of this metric over time we see that it characteristically will rise early in a program as risks are identified but there is a delay in mitigating them causing a rising metric as shown in Figure 38. As a program progresses, this value will rise to a peak at some point in the program and subsequently start a long decline. Risks continue to be identified so we see the accumulated total number of risks continue to increase but the program is being successful in mitigating risks and later risks are commonly lower in index than those identified earlier in the program.

Thus, we have arrived at a program risk index or metric. If we maintain a chart of this metric over time we see that it characteristically will rise early in a program as risks are identified but there is a delay in mitigating them causing a rising metric as shown in Figure 38. As a program progresses, this value will rise to a peak at some point in the program and subsequently start a long decline. Risks continue to be identified so we see the accumulated total number of risks continue to increase but the program is being successful in mitigating risks and later risks are commonly lower in index than those identified earlier in the program.

Figure 38  Program Risk Tracking Chart
3.5 Verification Requirements

The author respects the six-section specification structure of the DoD specification standard where Section 3 contains the product requirements and Section 4 the verification requirements. For every product requirement in Section 3 there should be one or more verification process requirements in Section 4. Whoever writes the Section 3 requirement should also write the verification requirement and with little time between the two actions. The rationale here is that the author of an unverifiable requirement will have some much difficulty writing a verification requirement and this difficulty may stimulate them to look for better ways of writing the requirement leading to a verifiable requirement.

We commonly respect four methods of verification: test, analysis, demonstration, and examination. A verification traceability matrix should be included in every specification that correlates every requirement in Section 3 with a method of verification and one or more verification requirements in Section 4. Each one of the rows in the verification traceability matrix forms a verification string. All of the verification traceability matrices are pooled into a single program-wide verification compliance matrix shown in Figure 39 that lists every verification string.

Figure 39 Verification Traceability
A verification engineer or team must now assign verification task numbers to all of the strings in the compliance matrix. Each task is identified in a task matrix and coordinated with the name of a principal engineer who must plan the task, make arrangements for the needed resources in a timely way relative to the task, accomplish the task on schedule, and produce a verification report. The reports from all of the tasks for a given item may be subjected to a configuration audit by the customer to ensure that the contractor did meet all of the requirements in the specification.

3.6 Specification Review and Approval

No matter the path the specification has taken through the requirements analysis process relative to modeling, the program should pass it through a review and approval process before it becomes a part of the formal configuration baseline definition. The review and approval process, shown in Figure 40, offers a formal or informal peer review way of comparing the content of a specification with a set of standards that all specifications should meet. Following approval, the specification must be formally released, published, and made available to program personnel either in paper or on-line form. The released specification must thereafter be formally protected through configuration management. Any changes must pass back through this same process to gain approval.

Figure 40 Specification Review and Approval

The formal review process should include a conscious evaluation of specification template faithfulness and overall quality measured in accordance with a specification checklist. Next, the specification should be checked for adherence with good traceability standards. The program may choose to fully implement traceability standards shown in the figure or some subset thereof. The final string of checks shown in Figure 40 assesses the specification for residual risk, completeness and excess content. A decision is reached by the reviewers followed by the review chairman calling either for corrective action or approval of the specification, if needed.

Specifications prepared on small or advanced programs may not have sufficient budget to support a formal review process. In this case, while not as supportive of a low risk approach, a
The master copy of each specification must be retained and protected by an assigned authority in order to protect the integrity of the document. Once approved and released, this master must be accurately identified and protected against change. In one organization the author recalls, the master was changed during work on an engineering change proposal (ECP) but the ECP was subsequently canceled. The organization no longer had a master for the specification in effect because it had become corrupted by the change work that did not materialize. It helps to consider each specification build or change a separate campaign that results in the release of a document that will exist forever. If subsequently that document is changed, the change is built anew on the preserved baseline past.

Specifications must be readily available to personnel working a program. As they are released they must be distributed to those who need them. As they are revised the same is true of the revisions. Years ago specifications were crafted with typewriters and type setting. These were published in paper form and distributed using shoe leather and mail systems. If most of your specifications are in paper media, you may have no near term option but to place them in a paper document library from which program personnel may check them out physically. But, even if this is the current case, you should be making plans to move to an on-line specification library for cost, efficiency, and document configuration control purposes.

In a paper media, after a specification is formally released, the master must go to reproduction where sufficient copies are made to cover the needs for distribution and the library. The master should be returned to what the author has accurately heard referred to as the vault, a physically secure facility (not in the classified data sense unless this is also a valid concern) where all of the engineering masters are retained. The copies must then be physically distributed. If the specification in question is also a customer-approved specification, another loop will be required to gain their approval prior to distribution. A networked library will avoid a great deal of this busy work.

Adios paper and good riddance. Even if you are currently using a paper media for distribution you probably already have the resources in place to convert to networked computer media. It requires specifications captured in computer file media, a network with adequate storage capacity and speed, and easy access from work stations on the part of the people. These features are present in most everyone’s shop today or are not beyond the pale to achieve. It is unimaginable that anyone would use a typewriter today to prepare a specification so they will always be created in some form of word processor or computer database. The results of this work can be passed on to the document release function on a disk or via the network connection and thereafter transferred to an on-line library from which anyone may open it but not change it.
AN EFFECTIVE SPECIFICATION DEVELOPMENT ALGORITHM TUTORIAL

SPECIFICATION READINESS

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, AGM-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
An Effective Specification Development Algorithm Tutorial Outline

| 2.1 | Specification Preparation Readiness |
| 2.2 | Specification Preparation Readiness |
| 2.3 | Traditional Structured Analysis |
| 2.4 | Traditional Structured Analysis |
| 2.5 | Unified Modeling Language |
| 2.6 | Unified Modeling Language |
| 2.7 | Requirements Management |
| 2.8 | Requirements Management |

Success Is Possible

- **The Goal**
  - Good specifications
  - On time
  - Affordable
- **The Plan**
  - A sound beginning - be prepared
  - A clear path to a successful state - always clear what must be done
  - An effective closing - a specification review and approval process
What is a Specification?

A specification contains all of the requirements for a given item.

The Word Requirement, From The Dictionary

- Something wanted or necessary.
- Something essential to the existence or occurrence of something else.
- A necessary characteristic or attribute of some thing (or item).
In Writing a Specification, What Is the Target?

How to Hit the Target of Minimized Completeness

- Every performance requirement traceable to a model from which it was derived
- Every external interface for the item identified and defined in interface requirements in the specification (unless ICD applied)
- Every specialty engineering discipline that has been mapped to the item is included in the specification
- Every environmental influence defined in the appropriate model (system, end item, component) mapped to appropriate specification content.
- Every requirement in the specification traceable to a parent item specification requirement (ideally applies to the system specification relative to user requirements as well).
- Requirements are quantified as appropriate to the statement.
- Requirements are validated (risks understood and mitigated).
Product Requirements Types

- Hardware
  - Performance
  - Constraints
    - Interface
    - Specialty Engineering
    - Environmental
- One view of software requirements
  - Functional
  - Non-Functional
  - Quality
- My view of software types
  - Same as systems and hardware

Requirements Types

All of these requirements must be identified before product and process detailed design work is started and they must be mutually consistent.
Attributes of a Good Requirement

- Achievable (validated)
- Quantified
- Achievable (validated)
- Verifiable/testable
- Unambiguous
- Complete (covers all cases)
- Performance specification
  - Design independent
  - What, not how
- Detail specification
  - Design dependent

Some Good Examples

Frequency coverage. Item frequency coverage shall be 225.0 to 399.9 Megahertz inclusive in tenth Megahertz steps.

Weight. Item weight shall be less than or equal to 240 pounds.

Range. Maximum achievable range shall be greater than or equal to 5,000 nautical miles while recognizing a fuel loading safety margin of 10% or more.

Range. Maximum range shall be greater than or equal to 2,500 nautical miles.

Reliability. Item MTBF shall be greater than or equal to 10,000 hours.
Some Bad Examples

- The screens used in the system shall be designed in a user friendly manner.
- Item weight shall not be greater than 153 pounds.
- Aircraft shall identify their position within 1000 feet of actual along and across track position using Loran C.
- Brakes shall function smoothly and stop the train in a safe distance.
- There shall be no hailstorms in the path of the aircraft.
- On most days, transmitter power output should be 100 watts.
- Go fast.
- Item shall work well and last a long time.
- Any favorites from your past?

Specifications Are Full of Sentences

1. THESE SENTENCES SHOULD BE WRITTEN IN THE SIMPLEST POSSIBLE WAY
2. THE SUBJECT IS THE ITEM CHARACTERISTIC ABOUT WHICH THE REQUIREMENT IS WRITTEN
3. VERB SHALL CLEARLY CALLS FOR COMPLIANCE

<table>
<thead>
<tr>
<th>SUBJECT</th>
<th>VERB</th>
<th>OBJECT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Item memory margin</td>
<td>shall be</td>
<td>greater than 100%</td>
</tr>
</tbody>
</table>
The Verb

• SHALL     Mandatory
• WILL      Contractor intends to perform
• SHOULD    Recommended, Desirable

The Subject

• Writing requirements is easy
• The difficult job is knowing what to write them **about** - the subject of the sentence
• That is why we model the problem space
Good Examples In Primitive Form

weight < 240 pounds
range > 5,000 nautical miles
MTBF > 10,000 hours

Requirements Analysis Strategies

MODELING

STRUCTURED DECOMPOSITION
ARCHITECTURE SYNTHESIS
ITEM IDENTIFICATION

CLONING

POWER GENERATING SYSTEM
POWER PLANT
COOLING SYSTEM
VALVE X
VALVE Y

CUSTOMER Q&A (ELICITATION)

FREESTYLE

LIKE ITEM
PARENT ITEM (FLOWDOWN)
STANDARD
COMPOUND STANDARD

LIKE ITEM
FREESTYLE IS FOR EXPERTS AND FOOLS

REQUIREMENTS ANALYSIS

FREESTYLE IS FOR EXPERTS AND FOOLS
A Foolproof Search For Subjects

Structured Modeling Tools

Specification Template

DID

ESSENTIAL CHARACTERISTICS

PRIMITIVE LIST

ITEM SPECIFICATION

SYSTEM DEFINITION DOCUMENT

LANGUAGE, STYLE, FORMAT

General Program Task-Resource Relationships
Work Product Development Suite
Specification Case

Document Progressions

TSA
TAILORED MIL-STD-961E
SYSTEM SPECIFICATION TEMPLATE
SYSTEM SPECIFICATION
EXHIBIT B-1

TSA
TAILORED MIL-STD-961E
HARDWARE ITEM PERFORMANCE SPECIFICATION TEMPLATE
HARDWARE ITEM PERFORMANCE SPECIFICATION
EXHIBIT B-2
Document Progressions

System Development Process Overview

Covered in this tutorial
Preparatory Steps

A Single Model Will Not Work

System Elements

Software Content

Hardware Content
Hardware and Systems Analysis Models

- Traditional structured analysis
  - Functional analysis
    - Functional flow diagramming
    - Enhanced functional flow diagramming (CORE)
    - Behavioral diagramming (RDD/IPO)
    - IDEF 0 (SADT)
    - Process flow analysis
    - Hierarchical functional analysis
- Constraints analysis
  - State diagramming
  - SysML

Computer Software Structured Analysis Models

- Process-oriented analysis
  - Flow charting
  - Modern Structured Analysis (Yourdon-Demarco)
  - Modern Structured Analysis (Hatley-Pirbhai)
- Data-oriented analysis
  - Table normalizing
  - IDEF-1X
- Object-oriented analysis
  - Early models
    - Unified Modeling Language (UML)
- DoD architecture framework (DoDAF)
Structured View of a Problem Space

Structured Analysis Methods
Comparison

<table>
<thead>
<tr>
<th>MULTI-FACETED APPROACHES</th>
<th>PRODUCT ENTITY FACET</th>
<th>FUNCTIONAL FACET</th>
<th>BEHAVIORAL FACET</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRADITIONAL STRUCTURED ANALYSIS</td>
<td>PRODUCT ENTITY BLOCK DIAGRAM</td>
<td>FUNCTIONAL FLOW DIAGRAM</td>
<td>SCHEMATIC BLOCK DIAGRAM</td>
</tr>
<tr>
<td>MODERN STRUCTURED ANALYSIS</td>
<td>HIERARCHICAL DIAGRAM</td>
<td>DATA FLOW DIAGRAM</td>
<td>P SPEC, STATE DIAGRAM</td>
</tr>
<tr>
<td>EARLY OBJECT-ORIENTED ANALYSIS</td>
<td>CLASS AND OBJECT DIAGRAM</td>
<td>DATA FLOW DIAGRAM</td>
<td>STATE DIAGRAM</td>
</tr>
<tr>
<td>UML</td>
<td>CLASS/OBJECT, COMPONENT, &amp; DEPLOYMENT DIAGRAMS</td>
<td>USE CASES AND ACTIVITY DIAGRAMS</td>
<td>STATE, SEQUENCE, AND COMMUNICATION DIAGRAMS</td>
</tr>
</tbody>
</table>

○ UNPRECEDENTED ANALYTICAL ENTRY FACET
Model Suggestions for Today

<table>
<thead>
<tr>
<th>SPECIFICATION TYPE</th>
<th>MODEL SUGGESTED</th>
</tr>
</thead>
<tbody>
<tr>
<td>System Specification</td>
<td>Traditional Structured Analysis</td>
</tr>
<tr>
<td>Hardware Performance Specification</td>
<td>Traditional Structured Analysis</td>
</tr>
<tr>
<td>Software Performance Specification, General</td>
<td>Unified Modeling Language (UML)</td>
</tr>
<tr>
<td>Software Performance Specification, Database</td>
<td>DoD-1X</td>
</tr>
<tr>
<td>Software Performance Specification, DoD IS</td>
<td>DoDAF</td>
</tr>
</tbody>
</table>

But, be prepared to move to the use of SysML coupled with UML and the eventual merge of the two into a more fully integrated common modeling method.

Preparatory Steps
Enabling Documentation

EDG defines the enterprise common process to be applied on all programs, the functional departments, and their process responsibilities which include preparing a department manual covering all common process work allocated to the department.

The system engineering department manual that defines the way system engineering will be applied on all programs including requirements analysis and specification management.

All functional departments should define templates and data item descriptions for all work products that are prepared as documents. One example of these artifacts is a set of specification data item descriptions that are coordinated with a particular template and a method of acquiring the content through modeling.

Preparatory Steps

[Diagram of Preparatory Steps]
MIL-STD-961E Template

1 Scope 3.1.19 Computer Resource Requirements
2 Applicable Documents 3.1.20 Logistics
3 Requirements 3.1.21 Personnel and Training
3.1 Functional and Performance Rqmts. 3.1.22 Requirements Traceability
3.1.1 Missions 3.2 Interface Requirements
3.1.2 Threat 3.2.1 GFP Interfaces
3.1.3 Required States and Modes 3.2.2 External Interface Requirements
3.1.4 Entity Capability Requirements 3.3 Design and Construction
3.1.5 Reliability 3.3.1 Production Drawings
3.1.6 Maintainability 3.3.2 Software Design
3.1.7 Deployability 3.3.3 Workmanship
3.1.8 Availability 3.3.4 Standards of Manufacture
3.1.9 Environmental Conditions 3.3.5 Process Definition
3.1.10 Transportability 3.3.6 Material Definition
3.1.11 Materials and Processes 3.4 Precedence and Criticality of Rqmts.
3.1.12 Electromagnetic Radiation 4 Verification
3.1.13 Nameplates and Product Markings 4.1 Methods of Verification
3.1.14 Productivity 4.2 Classes of Verification
3.1.15 Interchangeability 4.3 Inspections
3.1.16 Safety 5 Packaging
3.1.17 Human Factors Engineering 6 Notes
3.1.18 Security and Privacy

Recommended Template With Map for Traditional Structured Analysis

<table>
<thead>
<tr>
<th>PARA</th>
<th>TITLE</th>
<th>MODEL</th>
<th>DEPT</th>
<th>APP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>SCOPE</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>2.</td>
<td>APPLICABLE DOCUMENTS</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.</td>
<td>REQUIREMENTS</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1</td>
<td>Functional and performance requirements</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.1</td>
<td>Missions</td>
<td>Mission Analysis</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.2</td>
<td>Threat</td>
<td>Threat Analysis</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.3</td>
<td>Required states and modes</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
</tbody>
</table>
| 3.1.3.1 | Functional analysis | DID | 2100 | A
| 3.1.3.2 | Subordinate entities | DID | 2100 | C
| 3.1.3.3 | Interface relationships | DID | 2100 | D
| 3.1.3.4 | Specialty engineering requirements | DID | 2100 | E
| 3.1.3.5 | Environmental model | DID | 2100 | B
| 3.2 | Entry capability requirements | Functional Analysis | 2100 | A
| 3.2.m | Capability m | Functional Analysis | 2100 | A
| 3.2.m.n | Capability m, requirement n | Functional Analysis | 2100 | A
| 3.3 | Interface requirements | N-Square Diagram | 2100 | D
| 3.3.1.m | External interface m | N-Square Diagram | 2100 | D
| 3.3.1.m.n | External interface m, requirement n | N-Square Diagram | 2100 | D
| 3.3.2 | Internal interface requirements | N-Square Diagram | 2100 | D
| 3.3.2.m | Internal interface m | N-Square Diagram | 2100 | D
| 3.3.2.m.n | Internal Interface m, requirement n | N-Square Diagram | 2100 | D

A2-1-18
Recommended Template With Map for
Traditional Structured Analysis

<table>
<thead>
<tr>
<th>PARA</th>
<th>TITLE</th>
<th>MODEL</th>
<th>DEPT</th>
<th>APP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>SCOPE</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>2.</td>
<td>APPLICABLE DOCUMENTS</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.</td>
<td>REQUIREMENTS</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.1</td>
<td>Functional and performance requirements</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.2</td>
<td>Threat</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.3</td>
<td>Required states and modes</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.3.1</td>
<td>Functional analysis</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.3.2</td>
<td>Subordinate entities</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.3.3</td>
<td>Interface relationships</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.3.4</td>
<td>Specialty engineering requirements</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.3.5</td>
<td>Environmental model</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.2</td>
<td>Entity capability requirements</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.2.m</td>
<td>Capability m</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.2.m.n</td>
<td>Capability m, requirement n</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.3</td>
<td>Interface requirements</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.3.1.m</td>
<td>External interface requirements</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.3.1.m.n</td>
<td>External interface m, requirement n</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.3.2.1.m.n</td>
<td>Internal interface m, requirement n</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
</tbody>
</table>

Modeling Methods Map
Functional Department Map
Model Artifact Capture Map
Prepare and Maintain DIDs

- The DID follows the template format
- The recommended DID is focused on a particular modeling approach
- The DID tells how to create a specification using that template with a particular modeling approach

Here is a sample DID for a system specification using TSA.

Preparatory Steps
### Generic Specialty Engineering Scoping Matrix

<table>
<thead>
<tr>
<th>SPECIALTY DISCIPLINE</th>
<th>DEPT</th>
<th>PARA</th>
<th>A</th>
<th>A1</th>
<th>A2</th>
<th>A3</th>
<th>A4</th>
</tr>
</thead>
<tbody>
<tr>
<td>H1 Reliability</td>
<td>D2164</td>
<td>3.4.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H2 Maintainability</td>
<td>D2164</td>
<td>3.4.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H3 Availability</td>
<td>D2164</td>
<td>3.4.3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H4 Deployability and transportability</td>
<td>D2164</td>
<td>3.4.4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H5 Logistics</td>
<td>D2311</td>
<td>3.4.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H6 Maintenance</td>
<td>D2311</td>
<td>3.4.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H7 Interchangeability</td>
<td>D2131</td>
<td>3.4.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H8 Supply</td>
<td>D2311</td>
<td>3.4.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H9 Facilities and facility equipment</td>
<td>D2311</td>
<td>3.4.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HA Personnel</td>
<td>D2313</td>
<td>3.4.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HB Training</td>
<td>D2133</td>
<td>3.4.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HC Safety</td>
<td>D2105</td>
<td>3.4.6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HD Human factors engineering</td>
<td>D2105</td>
<td>3.4.7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HE Security and privacy</td>
<td>D2136</td>
<td>3.4.8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HF Electromagnetic compatibility</td>
<td>D2136</td>
<td>3.4.9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HG Lightning protection</td>
<td>D2136</td>
<td>3.4.10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HH Productibility</td>
<td>D2136</td>
<td>3.4.11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HI Affordability</td>
<td>D2108</td>
<td>3.4.12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HJ Computer resource requirements</td>
<td></td>
<td>3.4.13</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Design and construction</td>
<td></td>
<td>3.4.14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HK Quality Engineering</td>
<td>D2100</td>
<td>3.4.14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HL Parts, materials, and processes</td>
<td>D2167</td>
<td>3.4.14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Generic Specialty Engineering Scoping Matrix

<table>
<thead>
<tr>
<th>SPECIALTY DISCIPLINE</th>
<th>DEPT</th>
<th>PARA</th>
<th>A</th>
<th>A1</th>
<th>A2</th>
<th>A3</th>
<th>A4</th>
</tr>
</thead>
<tbody>
<tr>
<td>HM Workmanship</td>
<td>D5100</td>
<td>3.4.14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HN Nameplates and product markings</td>
<td>D2113</td>
<td>3.4.14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HO Serializatoin</td>
<td>D2113</td>
<td>3.4.14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HP Mass properties</td>
<td>D2124</td>
<td>3.4.14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HQ Structural properties</td>
<td>D2124</td>
<td>3.4.14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HR Shock and vibration</td>
<td></td>
<td>3.4.14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HS Earthquake survivability</td>
<td>D2123</td>
<td>3.4.14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HT Aerodynamics</td>
<td>D2144</td>
<td>3.4.14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HU Thermodynamics</td>
<td>D2143</td>
<td>3.4.14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HV Chemical, electrical, and mechanical properties</td>
<td></td>
<td>3.4.14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HW Stability</td>
<td></td>
<td>3.4.14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HX Coatings and Corrosion Control</td>
<td>D2167</td>
<td>3.4.14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Proposal Team Specification Actions

Environmental Requirements

- **System**
  - Identify spaces within which the system will have to function
  - Select standards covering those spaces
  - For each standard, select parameters that apply
  - Tailor the range of selected parameters

- **End item**
  - Build three dimensional model of end items, physical processes, and process environments
  - Extract item environments

- **Component**
  - Zone end item into spaces of common environmental characteristics
  - Map components to zones
  - Components inherit zone environmental requirements
Specialty Engineering Scoping Matrix
Applied to Program

Configuration Control the Models
On To Program Work

[Diagram of program work process]
AN EFFECTIVE SPECIFICATION DEVELOPMENT ALGORITHM TUTORIAL

TRADITIONAL STRUCTURED ANALYSIS

The Big Bang Theory Of System Development
THE TRADITIONAL APPROACH

EVERYTHING FLOWS FROM ONE IDEA,
THE CUSTOMER NEED
IT IS THE ULTIMATE REQUIREMENT, THE ULTIMATE FUNCTION

BA-BA-BA-BANG
Two Top-Level Views of a System

The Beginning Of Functional Decomposition
Traditional Structured Analysis Model Overview

The Ultimate Function and Its First Expansion

Alternative functional analysis techniques

Enhanced functional flow block diagramming (CORE)
Behavioral diagramming (RDD)
IDEF-0
Example of a Life Cycle Model

Use System Expansion Example
Space Transport System
Continued Function Decomposition

An orderly exposure of needed functionality moving from the known to the unknown, from simple to the complex, from the top to the bottom.

Performance Requirements Analysis and the RAS

<table>
<thead>
<tr>
<th>FUNCTION</th>
<th>PRODUCT ENTITY</th>
<th>REQUIREMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>MID</td>
<td>NAME</td>
<td>PID</td>
</tr>
<tr>
<td>F123</td>
<td>Provide Thrust</td>
<td>A12</td>
</tr>
</tbody>
</table>

Exposing what the system must do and how well it must do it encouraging identification of all essential characteristics and avoidance of unnecessary characteristics.
The Function-Product Entity Plane

Functional Analysis Alternatives

- IDEF 0
  - A variation on SADT
- Behavioral Diagramming
  - From Ascent Logic’s RDD
  - Based on IPO
- Enhanced Functional Flow Block Diagramming
  - Employed in Vitech’s CORE
- Hierarchical Functional Block Diagramming
Requirements Capture Using the RAS-Complete Format

Here Is What We Want

This Is How To Get It

Product Entity Structure

Use a common structure that includes hardware and software.
Systems Consist of Things and Relationships

Organizing For Interface Development

- Decompose needed functionality and allocate to product entities
- Map product entities to responsible development organizations
  - Create cross-functional integrated product and process teams (IPPT)
  - Assign principal engineers for lowest tier responsibilities on teams (everything has someone responsible)
- Establish clear rules for interface development responsibility
  - Identify needed interfaces as a function of how functionality was allocated to entities
  - Analyze product entity pair relationships using n-square diagrams
  - Partition interface into subsets as a function of product entity principal views
  - Assign interface responsibility to product entity principal engineers as a function of a receiving terminal rule (if you need an interface you must come forward)
  - System engineering manage the aggregate external and inter-team interface sets applying a lowest common team integration concept
- Minimize external (cross-organizational) interface at all levels, iterating product entity structure and/or development organization responsibilities to do so, if necessary, then apply system engineering integration resources to that which remains
Two Interface Definition Models

SCHEMATIC BLOCK DIAGRAMMING

- Lines define interfaces
- Blocks are objects only from the product entity structure

N-SQUARE DIAGRAMMING

- Marked intersections define interfaces
- Diagonal blocks are objects only from product entity block diagram
- Apparent ambiguity reflects directionality

Interface Requirements Derivation
Geometrical View
Development Often Fails at the Cross-organizational Interfaces

Interface Integration Focus
The Fundamental Problems in Interface Work

There is a one-to-one correspondence between teams and components. There is a one-to-two correspondence between teams and interfaces.

We tend to focus inwardly.
The Fundamental Problems in Interface Work

We are dependent on the worst interface on planet Earth in the development of interfaces.

Benefits Of Product Team Organization
Product Entity and Interface Responsibility

Lowest Common Team Concept
### Specialty Engineering Identification of Constraints

<table>
<thead>
<tr>
<th>Constraints</th>
<th>A11</th>
<th>A12</th>
<th>A13</th>
<th>A14</th>
<th>A15</th>
</tr>
</thead>
<tbody>
<tr>
<td>H1</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>H2</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H3</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H4</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H5</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>H6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>H7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>H8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>H9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>H10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

#### SPECIALTY ENGINEERING CONSTRAINTS MATRIX

#### SPECIALTY ENGINEERING REQUIREMENTS FLOW INTO THE INDICATED SPECIFICATIONS VIA THE RAS IMPLEMENTED IN A DATABASE

### Specialty Engineering Plane Added
Environment Subsets

Environmental Requirements

• System
  – Identify spaces within which the system will have to function
  – Select standards covering those spaces
  – For each standard, select parameters that apply
  – Tailor the range of selected parameters

• End item
  – Build three dimensional model of end items, physical processes, and process environments
  – Extract item environmental requirements

• Component
  – Zone end item into spaces of common environmental characteristics
  – Map components to zones
  – Components inherit zone environmental requirements
Environmental Planes Added

**RAS Complete In Tabular Form**

<table>
<thead>
<tr>
<th>Performance Requirements</th>
<th>Requirement Entry</th>
<th>Product Entity Requirement</th>
<th>Requirement Document Entity</th>
</tr>
</thead>
<tbody>
<tr>
<td>System Software</td>
<td>A</td>
<td>Product Systems</td>
<td>A</td>
</tr>
<tr>
<td>System Software</td>
<td>B</td>
<td>Product Systems</td>
<td>B</td>
</tr>
<tr>
<td>System Software</td>
<td>C</td>
<td>Product Systems</td>
<td>C</td>
</tr>
<tr>
<td>System Software</td>
<td>D</td>
<td>Product Systems</td>
<td>D</td>
</tr>
<tr>
<td>System Software</td>
<td>E</td>
<td>Product Systems</td>
<td>E</td>
</tr>
<tr>
<td>System Software</td>
<td>F</td>
<td>Product Systems</td>
<td>F</td>
</tr>
<tr>
<td>System Software</td>
<td>G</td>
<td>Product Systems</td>
<td>G</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>System Software</td>
<td>A</td>
<td>Product Systems</td>
<td>A</td>
</tr>
<tr>
<td>System Software</td>
<td>B</td>
<td>Product Systems</td>
<td>B</td>
</tr>
<tr>
<td>System Software</td>
<td>C</td>
<td>Product Systems</td>
<td>C</td>
</tr>
<tr>
<td>System Software</td>
<td>D</td>
<td>Product Systems</td>
<td>D</td>
</tr>
<tr>
<td>System Software</td>
<td>E</td>
<td>Product Systems</td>
<td>E</td>
</tr>
<tr>
<td>System Software</td>
<td>F</td>
<td>Product Systems</td>
<td>F</td>
</tr>
<tr>
<td>System Software</td>
<td>G</td>
<td>Product Systems</td>
<td>G</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Interface</th>
<th>Requirement Entry</th>
<th>Product Entity Requirement</th>
<th>Requirement Document Entity</th>
</tr>
</thead>
<tbody>
<tr>
<td>System Software</td>
<td>A</td>
<td>Product Systems</td>
<td>A</td>
</tr>
<tr>
<td>System Software</td>
<td>B</td>
<td>Product Systems</td>
<td>B</td>
</tr>
<tr>
<td>System Software</td>
<td>C</td>
<td>Product Systems</td>
<td>C</td>
</tr>
<tr>
<td>System Software</td>
<td>D</td>
<td>Product Systems</td>
<td>D</td>
</tr>
<tr>
<td>System Software</td>
<td>E</td>
<td>Product Systems</td>
<td>E</td>
</tr>
<tr>
<td>System Software</td>
<td>F</td>
<td>Product Systems</td>
<td>F</td>
</tr>
<tr>
<td>System Software</td>
<td>G</td>
<td>Product Systems</td>
<td>G</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Environmental</th>
<th>Requirement Entry</th>
<th>Product Entity Requirement</th>
<th>Requirement Document Entity</th>
</tr>
</thead>
<tbody>
<tr>
<td>System Software</td>
<td>A</td>
<td>Product Systems</td>
<td>A</td>
</tr>
<tr>
<td>System Software</td>
<td>B</td>
<td>Product Systems</td>
<td>B</td>
</tr>
<tr>
<td>System Software</td>
<td>C</td>
<td>Product Systems</td>
<td>C</td>
</tr>
<tr>
<td>System Software</td>
<td>D</td>
<td>Product Systems</td>
<td>D</td>
</tr>
<tr>
<td>System Software</td>
<td>E</td>
<td>Product Systems</td>
<td>E</td>
</tr>
<tr>
<td>System Software</td>
<td>F</td>
<td>Product Systems</td>
<td>F</td>
</tr>
<tr>
<td>System Software</td>
<td>G</td>
<td>Product Systems</td>
<td>G</td>
</tr>
</tbody>
</table>
Save the Models!
UNIFIED MODELING LANGUAGE

Agenda

• The Dead Sea scrolls of software development
• UML modeling artifacts
• UML modeling approach
• Integration
• Requirements and modeling documentation
Software Dead Sea Scrolls

- Flow chartings
  - Functionality examined
  - Data in the back seat
- SADT and IPO
  - Two axis models
- Modern structured analysis and HP
  - Functionality and data examined
- Early OOA
  - Search for objects and their behavior
  - Anti Sullivan

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 tutorial - form follows function.

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

UML can support either direction.

Note: A classifier is a general term for a software product entity represented by a node, component, or class in UML.
The Software Development Process

- Identify a product entity that will be developed as computer software.
- Dynamically analyze the entity.
  - Use cases
  - Sequence diagram
  - Communication diagram
  - Activity diagram
  - State diagram
- In the sequence, communication, and activity diagramming analysis you will have to identify lower tier 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.
Suggested SRS Structure

3 REQUISITES
3.1 Required states and modes
3.2 Software entity capability requirements
3.2.m Software entity capability m
3.2.m.n Software entity capability m, requirement n
3.3 Software entity interface requirements
3.3.1 Software entity external interface requirements
3.3.1.m Specific external interface m
3.3.1.m.n External interface m, requirement n
3.3.2 Software entity internal interface requirements
3.3.2.m Specific internal interface m
3.3.2.m.n Internal interface m, requirement n
3.3.3 Software entity internal data requirements
3.3.3.n Specific software entity internal data requirement n
3.4 Specialty engineering requirements
3.5 Software entity environmental requirements
3.6 Precedence and criticality requirements

The Diagrams of UML

- For modeling dynamic aspects of the system
  - Use case diagram
  - Sequence diagram
  - Timing diagram
  - Communication diagram (renamed in 2)
  - State diagram
  - Activity 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)

(2) = added in UML 2.0
The Dynamic Models

Sequence Diagram UX11321

Activity Diagram UX11323

State Diagram UX11324

Interaction Diagrams

Communication Diagram UX11322

Sequence Diagram UX-11321

Emphasizes the time ordering of messages

Actor

Classifier AX1

Classifier AX2

messageOne()

messageTwo()

messageFour()

messageThree()

Argument List

Lifeline active

Time

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

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

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
A Simple Example

Communication Diagram UX11322
Emphasizes structural relationships

Semantically identical to the sequence diagram.
Activity Diagram UX11323

State Diagram UX11324
The Static Entities in UML

- **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 node or collection of collection of nodes.

- **Node**
  - Appears on a deployment diagram that exists at run time and 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 classes.

- **Class**
  - A description of a set of objects that share the same attributes, operations, relationships, and semantics.

- **Object**
  - An instance of a class.

UML Structural Artifacts in a Product Entity Structure
Deployment and Component Diagrams

Classes and Objects

A class is a description of a set of objects that share the same attributes, operations, relationships, and semantics. An object is an instance of a class. Graphically a class is rendered as a rectangle.

<table>
<thead>
<tr>
<th>NameName</th>
<th>The name is a noun or noun phrase</th>
</tr>
</thead>
<tbody>
<tr>
<td>attributeOne</td>
<td>An attribute is a named property of a class that describes a range of values that instances of the property may hold. An attribute represents some property of the thing you are modeling that is shared by all objects of that class.</td>
</tr>
<tr>
<td>attributeTwo</td>
<td></td>
</tr>
<tr>
<td>attributeThree</td>
<td></td>
</tr>
<tr>
<td>attributeFour</td>
<td></td>
</tr>
<tr>
<td>operationOne()</td>
<td>An operation is the implementation of a service that can be requested from any object of a class to effect behavior. An operation is an abstraction of something you can do to an object that is shared by all objects of that class.</td>
</tr>
<tr>
<td>operationTwo()</td>
<td></td>
</tr>
<tr>
<td>operationThree()</td>
<td></td>
</tr>
</tbody>
</table>
Class Responsibilities

A responsibility is a contract or an obligation of a class. You may find it useful to begin the analysis of classes this way translating these into attributes and operations that best fulfill the class’s responsibility as the model is refined. The responsibility is noted in an added compartment in which descriptive free form-text is entered.

Structural Relationships
These Have Nothing To Do With Messages

NameOne is the base class. NameTwo and NameThree are leaf classes in a generalization. NameOne depending on NameFour for information and services. An association is a structural relationship.
Structural Relationships
Association Adornments

- Association Name
  - NameOne
  - x-y
  - Association name
  - role
  - NameTwo
  - role
  - name direction

- Association Role
  The face that the class at the far end of an association presents to the class at near end of the association. Role names called end names.

- Association Multiplicity
  Tells how many objects may be connected across an association instance. Given by a range of numbers.

- Association Aggregation
  Expresses a whole-part relationship between to associated classes.

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 use case.
- For each scenario build a sequence diagram and in the process identify next lower tier classifiers and messages between the actors and lower tier classifiers.
- Apply communication, activity, and state diagrams as needed.
- Derive requirements from dynamic modeling artifacts.

Hierarchical Modeling Relationships
Context Diagram

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

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

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

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

Use Case Fundamentals

- A use case is a more expressive context diagram common 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

Actors derive tangible benefits from the system.

- **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 UX-11

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.
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.

Examine Each Scenario Dynamically

- Activity, sequence, and communication diagrams require identification of lower tier entities leading to additional of entities on the consolidated product entity diagram
- State diagrams may also be useful in identifying essential characteristics appropriate for the entity being analyzed
- Requirements flow out of the dynamic analysis and into the specification for the entity being analyzed
AN EFFECTIVE SPECIFICATION DEVELOPMENT ALGORITHM TUTORIAL

REQUIREMENTS MANAGEMENT

Agenda

• Process summary
• Organizing and specification responsibility
• Requirements risk management
• Traceability
• Tools
• Review, approval, release, and distribution
• A peek into the future
Life Cycle Model

Suggested Enterprise Structure
Benefits of Product-Oriented Team Structure

The System Product Entity Structure and Teaming
Risk Defined

- The danger that injury, damage, or loss will occur
- Somebody or something likely to cause injury, damage, or loss
- The probability, amount, or type of possible loss incurred by an insurer
- The possibility of loss in an investment or speculation
- The statistical chance of danger from something, especially from the failure of an engineered system

Risk Measurement Parameters

### Probability of Occurrence

<table>
<thead>
<tr>
<th>CAT</th>
<th>TITLE</th>
<th>P(O)</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Nearly Certain</td>
<td>0.95-1.00</td>
<td>Will occur at least once during program</td>
</tr>
<tr>
<td>4</td>
<td>Probable</td>
<td>0.75-0.95</td>
<td>Will probably occur once during program</td>
</tr>
<tr>
<td>3</td>
<td>Possible</td>
<td>0.50-0.75</td>
<td>May occur during program</td>
</tr>
<tr>
<td>2</td>
<td>Unlikely</td>
<td>0.25-0.49</td>
<td>Will probably not occur during program</td>
</tr>
<tr>
<td>1</td>
<td>Nearly Impossible</td>
<td>0.00-0.24</td>
<td>Will not occur during program</td>
</tr>
</tbody>
</table>

### Seriousness of Effect

<table>
<thead>
<tr>
<th>CAT</th>
<th>TITLE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Catastrophic</td>
<td>Program in jeopardy of termination</td>
</tr>
<tr>
<td>4</td>
<td>Serious</td>
<td>Serious damage to program</td>
</tr>
<tr>
<td>3</td>
<td>Moderate</td>
<td>Problems cause program focus difficulties</td>
</tr>
<tr>
<td>2</td>
<td>Minor</td>
<td>Problems that can be easily be overcome</td>
</tr>
<tr>
<td>1</td>
<td>Null</td>
<td>No problem</td>
</tr>
</tbody>
</table>
### Risk Metric Values

<table>
<thead>
<tr>
<th>SERIOUSNESS OF THE EFFECT</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>25</td>
<td>20</td>
<td>15</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>20</td>
<td>16</td>
<td>12</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>15</td>
<td>12</td>
<td>9</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>8</td>
<td>6</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

Legend:
- **High Risk**
- **Medium Risk**
- **Low Risk**

### Aggregate Program Risk Index

![Graph showing Aggregate Program Risk Index](image)

Legend:
- **Program Risk Index**
- **Number of Risks**

**Peak Risk Point**
Traceability Forms

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

Traceability Integration

- Lateral traceability within TSA and UML independently should be no problem with all requirements derived from models
- Vertical interface traceability is decomposition driven within TSA and UML as well as across the HW-SW gap
- Environmental requirements are significantly different between HW and SW so traceability is not a significant issue between them
- Performance requirements across the HW-SW gap offer a vertical traceability challenge
The System Product Entity Structure

This is the kind of relationship of interest

Level at which a subordinate software entity is identified

Hardware entity
Software entity

System

The System Product Entity Structure

Traceability Across the Gap

The Gap

• 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

Alternatively, one could rely upon experienced inspection without the organizing influence of the matrix.
Today's Tools

Tools Integration
Tomorrow’s Tools

- Front end modeling tools
  - Use case modeling
  - Function/activity modeling
  - State modeling (behavioral modeling)
  - Sequence/timeline modeling
  - Product entity and interface modeling
  - Specialty engineering database linkage
  - Environmental coverage
- Connection of modeling to management database
- Big dumb database
  - Requirements capture
  - Traceability
  - Value management
  - Specification publishing

Tools Integration

- Traditional structured analysis
- Data base loaders
- Data base systems
- Data base loaders
- UML
- Data base MGMT
Integrated Tool Set

Configuration Management of Requirements Documentation

- REQUIREMENTS ANALYSIS
- DATABASE CONTENT
- COMPUTER PROJECTION REVIEW
- SYSTEM ENGINEERING MANAGED CONTENT
- CONFIGURATION MANAGED CONTENT
- PUBLISH AND RELEASE
- LIBRARY

Portion of database corresponding to released specifications and library content under formal configuration control.
Utility Of Computer Projection

- ON-LINE NETWORK CAPABILITY
- PUT THE PROJECTION CAPABILITY IN THE WORK AREA
- APPLY REAL-TIME CONCURRENT DEVELOPMENT (IPD)
- FORM AND REFORM BETWEEN MEETING AND INDIVIDUAL WORK QUICKLY

Specification Review and Approval Process
Evaluate for Template Faithfulness

- Compare specification cover data with template (standard)
- Compare specification paragraphing structure with template
- Compare specification style with template style guide

Individual Requirement Quality

- Spot check specification requirements for requirements quality checklist compliance
- Spot check specification for requirements quantification where appropriate
Section 2 Traceability

- All documents listed in Section 2 called somewhere in the specification
- All documents tailored, if necessary, to limit coverage to the application
- All documents called in the requirements listed in Section 2
- Spot check for excessively tailored standards which could be quoted instead of being called
- Ensure documents called are current and accepted authorities for the application

Completeness and Avoidance of Unnecessary Content

- Ask principal engineer how content was derived
  - If ad hoc, there should be concern
  - If through structured analysis, spot check how a few requirements were derived (ask to see the supporting modeling data)
- Ensure all requirements traceable to parent requirements
Residual Risk Evaluation

- All TBD/TBR are closed or, if not, are being carried as program or team risks
- An approved concept exists
- 
- 
- 

Movement To Universal Method

[Diagram of Movement To Universal Method]

- UNPRECEDENTED SYSTEM DEVELOPMENT
- PRECEDENTED SYSTEM DEVELOPMENT
UML and Functional Analysis

Unified Modeling Language (UML)

<table>
<thead>
<tr>
<th>STATIC DIAGRAMS</th>
<th>DYNAMIC DIAGRAMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEPLOYMENT DIAGRAM</td>
<td>COMPONENT DIAGRAM</td>
</tr>
<tr>
<td>OBJECT &amp; CLASS DIAGRAMS</td>
<td>STATE CHART</td>
</tr>
<tr>
<td>ARCHITECTURE BLOCK DIAGRAM</td>
<td>COMMUNICATION DIAGRAM</td>
</tr>
<tr>
<td>SEQUENCE DIAGRAM</td>
<td>USE CASE DIAGRAM</td>
</tr>
<tr>
<td>FUNCTIONAL FLOW DIAGRAM</td>
<td></td>
</tr>
</tbody>
</table>

Traditional Structured Analysis
A Subset of UML?

Modeling Changes In the Near Term
System Modeling Evolution Timeline

Over the Hill and Through the Woods to Utopia
Review and Summary

The target is completeness and avoidance of unnecessary content

Use models to identify essential characteristics

Do the analysis
Write requirements in primitive form based on essential characteristics identified through modeling.

Translate into full sentences and insert into paragraph structure.
Practical Six Sigma Tools for Systems Engineering

9th Annual Systems Engineering Conference
23-26 October 2006

Rick Hefner, Ph.D.
Director, Process Management
Northrop Grumman Corporation
Background

- Six Sigma has proven to be a powerful enabler for process improvement
  - CMMI adoption
  - Process improvement for measurable ROI
  - Statistical analysis
- This presentation will focus on practical tools and techniques for use by systems engineers

Agenda

- What is Six Sigma?
- How does it apply to systems engineering?
- Strategies and lessons learned
Projects Have Historically Suffered from Mistakes

People-Related Mistakes
1. Undermined motivation
2. Weak personnel
3. Uncontrolled problem employees
4. Heroics
5. Adding people to a late project
6. Noisy, crowded offices
7. Friction between developers and customers
8. Unrealistic expectations
9. Lack of effective project sponsorship
10. Lack of stakeholder buy-in
11. Lack of user input
12. Politics placed over substance
13. Wishful thinking

Process-Related Mistakes
14. Overly optimistic schedules
15. Insufficient Risk Management
16. Contractor failure Insufficient planning
17. Abandonment of planning under pressure
18. Wasted time during the fuzzy front end
19. Shortchanged upstream activities
20. Inadequate design
21. Shortchanged quality assurance
22. Insufficient management controls
23. Premature or too frequent convergence
24. Omitting necessary tasks from estimates
25. Planning to catch up later
26. Code-like-hell programming

Product-Related Mistakes
28. Requirements gold-plating
29. Feature creep
30. Developer gold-plating
31. Push me, pull me negotiation
32. Research-oriented development

Technology-Related Mistakes
33. Silver-bullet syndrome
34. Overestimated savings from new tools or methods
35. Switching tools in the middle of a project
36. Lack of automated source-code control

Reference: Steve McConnell, Rapid Development

Standish Group, 2003 survey of 13,000 projects
- 34% successes
- 15% failures
- 51% overruns
Many Approaches to Solving the Problems

- Which weaknesses are causing my problems?
- Which strengths may mitigate my problems?
- Which improvement investments offer the best return?
Approaches to Process Improvement

**Data-Driven (e.g., Six Sigma, Lean)**
- Clarify what your customer wants (Voice of Customer)
  - Critical to Quality (CTQs)
- Determine what your processes can do (Voice of Process)
  - Statistical Process Control
- Identify and prioritize improvement opportunities
  - Causal analysis of data
- Determine where your customers/competitors are going (Voice of Business)
  - Design for Six Sigma

**Model-Driven (e.g., CMM, CMMI)**
- Determine the industry best practice
  - Benchmarking, models
- Compare your current practices to the model
  - Appraisal, education
- Identify and prioritize improvement opportunities
  - Implementation
  - Institutionalization
- Look for ways to optimize the processes
What is Six Sigma?

- Six Sigma is a management philosophy based on meeting business objectives by reducing variation
  - A disciplined, data-driven methodology for decision making and process improvement
- To increase process performance, you have to decrease variation

- Greater predictability in the process
- Less waste and rework, which lowers costs
- Products and services that perform better and last longer
- Happier customers

**Diagram:**

- Too early vs. Too late in delivery time with defects.
- Spread of variation too wide compared to specifications.
- Reduce variation to make spread narrow compared to specifications.

**Text:**

- Defects
- Delivery Time
- Reduce variation

**Graph:**

- Too early, Too late, Delivery Time, Defects
- Spread of variation too wide compared to specifications.
- Reduce variation, Spread of variation narrow compared to specifications.
A Typical Six Sigma Project in Systems Engineering

- The organization notes that systems integration has been problematic on past projects (budget/schedule overruns).
- A Six Sigma team is formed to scope the problem, collect data from past projects, and determine the root cause(s).
- The team’s analysis of the historical data indicates that poorly understood interface requirements account for 90% of the overruns.
- Procedures and criteria for a peer review of the interface requirements are written, using best practices from past projects.
- A pilot project uses the new peer review procedures and criteria, and collects data to verify that they solve the problem.
- The organization’s standard SE process and training is modified to incorporate the procedures and criteria, to prevent similar problems on future projects.
Roles & Responsibilities - Organizational Implementation

- **Champions** – Facilitate the leadership, implementation, and deployment
- **Sponsors** – Provide resources
- **Process Owners** – Responsible for the processes being improved
- **Master Black Belts** – Serve as mentors for Black Belts
- **Black Belts** – Lead Six Sigma projects
  - Requires 4 weeks of training
- **Green Belts** – Serve on improvement teams under a Black Belt
  - Requires 2 weeks of training
Applicability to Engineering

- **System engineering processes are fuzzy**
  - Systems engineering "parts" are produced using processes lacking predictable mechanizations assumed for manufacturing of physical parts
  - Simple variation in human cognitive processes can prevent rigorous application of the Six Sigma methodology
  - Process variation can never be eliminated or may not even reduced below a moderate level

- **Results often cannot be measured in clear $ savings returned to organization**
  - Value is seen in reduced risk, increased customer satisfaction, more competitive bids, …
How Six Sigma Helps Process Improvement

- PI efforts often generate have little direct impact on the business goals
  - Confuses ends with means; results measured in activities implemented, not results
- Six Sigma delivers results that matter to managers (fewer defects, higher efficiency, cost savings, ...)
- Six Sigma concentrates on problem solving in small groups, focused on a narrow issue
  - Allows for frequent successes (3-9 months)
- Six Sigma focuses on the customer’s perception of quality
How Six Sigma Helps CMMI-Based Improvement

- For an individual process:
  - CMM/CMMI identifies what activities are expected in the process
  - Six Sigma identifies how they can be improved (efficient, effective)

<table>
<thead>
<tr>
<th>SG 1 Establish Estimates</th>
</tr>
</thead>
<tbody>
<tr>
<td>SP 1.1 Estimate the Scope of the Project</td>
</tr>
<tr>
<td>SP 1.2 Establish Estimates of Project Attributes</td>
</tr>
<tr>
<td>SP 1.3 Define Project Life Cycle</td>
</tr>
<tr>
<td>SP 1.4 Determine Estimates of Effort and Cost</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SG 2 Develop a Project Plan</th>
</tr>
</thead>
<tbody>
<tr>
<td>SP 2.1 Establish the Budget and Schedule</td>
</tr>
<tr>
<td>SP 2.2 Identify Project Risks</td>
</tr>
<tr>
<td>SP 2.3 Plan for Data Management</td>
</tr>
<tr>
<td>SP 2.4 Plan for Project Resources</td>
</tr>
<tr>
<td>SP 2.5 Plan for Needed Knowledge and Skills</td>
</tr>
<tr>
<td>SP 2.6 Plan Stakeholder Involvement</td>
</tr>
<tr>
<td>SP 2.7 Establish the Project Plan</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SG 3 Obtain Commitment to the Plan</th>
</tr>
</thead>
<tbody>
<tr>
<td>SP 3.1 Review Subordinate Plans</td>
</tr>
<tr>
<td>SP 3.2 Reconcile Work and Resource Levels</td>
</tr>
<tr>
<td>SP 3.3 Obtain Plan Commitment</td>
</tr>
</tbody>
</table>

Example – Project Planning in the CMMI

- Could fully meet the CMMI goals and practices, but still write poor plans
- Six Sigma can be used to improve the planning process and write better plans
How CMMI Helps Six Sigma Based Improvement

- CMM/CMMI focuses on organizational change
  - Provides guidance on many dimensions of the infrastructure

### Process Areas
Organizational Process Focus
Organizational Process Definition
Organizational Training
Organizational Process Performance
Organizational Innovation and Deployment

### Generic Practices (all process areas)
- GP 2.1 Establish an Organizational Policy
- GP 2.2 Plan the Process
- GP 2.3 Provide Resources
- GP 2.4 Assign Responsibility
- GP 2.5 Train People
- GP 3.1 Establish a Defined Process
- GP 2.6 Manage Configurations
- GP 2.7 Identify and Involve Relevant Stakeholders
- GP 2.8 Monitor and Control the Process
- GP 3.2 Collect Improvement Information
- GP 2.9 Objectively Evaluate Adherence
- GP 2.10 Review Status with Higher-Level Management
Barriers and Challenges

- Capturing the first, “low hanging fruit” makes Six Sigma implementation look easy…
  - Clearer problems, simpler solutions, bigger payoffs
  - Little need for coordination

  …but later projects are tougher
  - Keeping projects appraised of similar efforts, past and current
  - Focusing on “the pain”, not the assumed solution

- Engineering process measurements are often difficult to analyze
  - Dirty (or no) data, human recording problems
  - May necessitate Define-Measure-Analyze-Measure-Analyze-etc.

- Must demonstrate the value of quantitative data to managers
  - Management style - reactive vs. proactive vs. quantitative
  - Less value in a chaotic environment
  - Must engage customers
Additional Challenges

- **Difficulty in collecting subjective, reliable data**
  - Humans are prone to errors and can bias data
  - E.g., the time spent in privately reviewing a document

- **Dynamic nature of an on-going project**
  - Changes in schedule, budget, personnel, etc. corrupt data

- **Analysis requires that complex SE processes be broken down into small, repeatable tasks**
  - E.g., peer review

- **Repeateable process data requires the project/organization to define (and follow) a detailed process**
Tools & Techniques
DMAIC –
A Structured Approach to Improving a Process

1. DEFINE
2. MEASURE
3. ANALYZE
4. IMPROVE
5. CONTROL
DMAIC - Define

- Purpose is to set project goals and boundaries
- Establishes upfront focus on customer
- Key products
  - Project charter
  - Process map
  - List of what is important to customer -- Critical to Quality factors (CTQs)
Identify Key Stakeholders Early On

- Develop communication plan based on level of commitment required

<table>
<thead>
<tr>
<th>Level of Commitment</th>
<th>Testers</th>
<th>Developers</th>
<th>Requirements Leads</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enthusiastic Support</td>
<td></td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>Help it work</td>
<td>O</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Compliant</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hesitant</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Indifferent</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Uncooperative</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Opposed</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hostile</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Stakeholders (Examples)

X = Current Level of Commitment
O = Level of Commitment Necessary for Success
Suppliers-Inputs-Process-Outputs-Customers (SIPOC)

- High level “as-is” process map
  - 5 to 7 key steps of main action
  - Used to focus on the fundamental elements of the process
Voice of the Customer

- CT "critical to" matrix links process or CT tree (columns of the matrix) and product or CTY tree (rows)
  - Critical To Satisfaction (CTS)
  - Critical To Quality (CTQ)
  - Critical To Delivery (CTD)
  - Critical To Cost (CTC)
  - Critical To Process (CTP) - Process parameters which significantly influence a CTQ, CTD, and/or CTC
DMAIC - Measure

- Purpose is to narrow range of potential causes and establish a baseline capability level
  - Identify specific problem(s)
  - Prioritize critical input/process/output measures
  - Validate measurement system
- Key products
  - Cause/effect diagrams
  - FMEA
  - Gage R&R
  - Data collection plan
  - Analysis results
Failure Modes and Effects Analysis

- Used to identify the way in which errors happen; an error mode, the antithesis of function
- Employed as a diagnostic tool in *improvement*
- Used as a prevention tool in *design*
- Deals with the three dimensions of an error mode:  
  - Severity  
  - Detectability  
  - Frequency

<table>
<thead>
<tr>
<th>Process/Product Failure Modes and Effects Analysis (FMEA)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process/Part Number</td>
</tr>
<tr>
<td>----------------------------------------------------------</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

Copyright 2005 Northrop Grumman Corporation
Data Collection Plan

### Measurement Consistency and Accuracy

#### Data Collection Plan

<table>
<thead>
<tr>
<th>What questions do you want to answer?</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Data</strong></td>
</tr>
<tr>
<td>What Measure type/Data type</td>
</tr>
<tr>
<td>How will you ensure consistency and stability?</td>
</tr>
</tbody>
</table>

- **Meaning of measurement in relation to project, process, or product**
- **What is counted into or excluded**
- **Measurement validation method**
  - Regular validation during collection
  - Periodic validation of samples or aggregates independent of collection tools and methods
- **Frequency of measurement collection**
- **Calculations used to derive an indirect, aggregated, or accumulated value**
- **How and where measurements are stored and accessed**
- **Tools, methods, resources, and assignments required**
DMAIC - Analyze

- Purpose is to evaluate data/information for trends, patterns, causal relationships and "root causes"

- Key products
  - Quantitative analysis results
  - Theory that has been tested
Six Sigma Tool Kit

Stratification

Queue 1

Queue 2

Hypothesis Testing

Chi-Square \( \chi^2 \)  t-test ANOVA

Regression

Capability Analysis

Process Sigma = 2.7

LSL  USL

Regression Analysis

Control Charts

UCL  LCL

Boxplot ANOVAs

Data Analysis

Copyright 2005 Northrop Grumman Corporation
Exercise – What is Quantitative Management?

- Suppose your project conducted several peer reviews of similar code, and analyzed the results:
  - Mean = 7.8 defects/KSLOC
  - +3σ = 11.60 defects/KSLOC
  - -3σ = 4.001 defects/KSLOC

- What would you expect the next peer review to produce in terms of defects/KSLOC?
- What would you think if a review resulted in 10 defects/KSLOC?
- 3 defects/KSLOC?
Exercise – What is Required for Quantitative Management?

- What is needed to develop the statistical characterization of a process?
  - The process has to be stable (predictable)
    - Process must be consistently performed
    - Complex processes may need to be stratified (separated into simpler processes)
  - There has to be enough data points to statistically characterize the process
    - Processes must occur frequently within a similar context (project or organization)

[Graph showing statistical data with mean, UCL, and LCL values]
What Is a Control Chart?

- A time-ordered plot of process data points with a centerline based on the average and control limits that bound the expected range of variation.
- Control charts are one of the most useful quantitative tools for understanding variation.
What Are the Key Features of a Control Chart?

Observation Number

Mean = 7.8

UCL = 11.60

LCL = 4.001

Upper Control Limit

Lower Control Limit

Process “Average”

Individual data points

Time ordered x-axis
There are Many Types of Control Charts

Tests performed with unequal sample sizes

U Chart of Defect Detected in Requirements Definition

- UCL = 0.09633
- LCL = 0.06017
- $\bar{U} = 0.07825$

Defects per Line of Code

ID

1 3 5 7 9 11 13 15 17 19 21 23 25
What is *Special Cause* and *Common Cause* Variation?

### Common Cause Variation
- **Routine** variation that comes from within the process
- Caused by the natural variation in the process
- Predictable (stable) within a range

### Special Cause Variation
- **Assignable** variation that comes from outside the process
- Caused by a unexpected variation in the process
- Unpredictable

**Observation Number** vs. **Individual Value**
- **Mean** = 7.8
- **UCL** = 11.60
- **LCL** = 4.001

**Observation Number** vs. **Individual Value**
- **Mean** = 7.467
- **UCL** = 13.36
- **LCL** = 1.578
What Is a *Stable* (Predictable) Process?

**U Chart of Defects Detected in Requirements Definition**

All data points within the control limits. No signals of special cause variation.

- **UCL** = 0.09633
- **LCL** = 0.06017
- **U** = 0.07825
What if the Process Isn’t Stable?

- You may be able to explain out of limit points by observing that they are due to an variation in the process
  - E.g., peer review held on Friday afternoon
  - You can eliminate the points from the data, if they are not part of the process you are trying to predict

- You may be able to stratify the data by an attribute of the process or attribute of the corresponding work product
  - E.g., different styles of peer reviews, peer reviews of different types of work products
Hearing Voices

- **Voice of the process**
  = the natural bounds of process performance

- **Voice of the customer**
  = the goals established for the product/process performance

- **Voice of the business**
  = process performance needed to be competitive

- **Process capability may be determined for the**
  - Organization
  - Product line
  - Project
  - Individual

- **Typically, the higher the level of analysis, the greater the variation**
Common Challenges for Engineering

- Data are often discrete rather than continuous, e.g., defects
- Observations often are scarce
- Processes are aperiodic
- Size of the object often varies, e.g., software module
- Data distributions may not be normal
How Do I Address These Challenges?

- Employ control chart types that specifically deal with discrete data distributions, e.g., $u$-charts and $p$-charts
- Use control charts that compensate for widely variable areas of opportunity
- Transform non-normal continuous data to normal data before constructing a control chart
- Cross check control charts with hypothesis tests where few data points exist
Typical Choices in Industry

- Most customers care about:
  - Delivered defects
  - Cost and schedule

- So organizations try to predict:
  - Defects found throughout the lifecycle
  - Effectiveness of peer reviews, testing
  - Cost achieved/actual (Cost Performance Index – CPI)
  - Schedule achieved/actual (Schedule Performance Index – SPI)

**Defect Detection Profile**

**Process performance**
- **Process measures** (e.g., effectiveness, efficiency, speed)
- **Product measures** (e.g., quality, defect density).
How Can Quantitative Management Help?

- By measuring both the mean and variation, the project/organization can assess the full impact of an “improvement”

- Can focus on reducing the variation (making the process more predictable)
  - Train people on the process
  - Create procedures/checklists
  - Strengthen process audits

- Can focus on increasing the mean (e.g., increase effectiveness, efficiency, etc.)
  - Train people
  - Create checklists
  - Reduce waste and re-work
  - Replicate best practices from other projects

- Can do both
DMAIC – Improve

- **Purpose is to develop, implement and evaluate solutions targeted at identified root causes**

- **Key products**
  - Candidate solutions
  - Pilot results
  - Risk assessment
  - Implementation plan
DMAIC - Control

- Purpose is to make sure problem stays fixed and new methods can be further improved over time

- Products
  - Control plan
  - Process documentation
  - Key learnings
The Control Plan

- Systematic tool for identifying and correcting root causes of out-of-control conditions
- Focus is on prevention -- not a “triggering” system
- Questions that must be answered
  - Who owns the process?
  - How will we transition responsibility from the Black Belt to the process owner?
  - How do we ensure we maintain the gains?
  - How do we track our results (performance and financial)?
Institutionalize Key Learnings

- Capture knowledge gained from Six Sigma project
  - Results
  - Key learnings
  - Potential future projects

- Communicate to rest of organization for knowledge sharing and transfer

- Archive in knowledge management repositories
Lessons Learned
Mission Success Requires Multiple Approaches

- Risk Management
- Systems Engineering
- Independent Reviews
- Training, Tools, & Templates
- Program Effectiveness
- Mission Assurance
- Operations Effectiveness
- Dashboards for Enterprise-Wide Measurement
- Communications & Best-Practice Sharing
- Robust Governance Model (Policies, Processes, Procedures)
- CMMI Level 5 for Software, Systems, and Services
- ISO 9001 and AS-9100 Certification
- Six Sigma
Benefits

Based on 18 Northrop Grumman CMMI Level 5 organizations

- Having multiple improvement initiatives helps encourage a change in behavior as opposed to “achieving a level”
  - Reinforces that change (improvement) is a way of life
- The real ROI comes in institutionalizing local improvements across the wider organization
  - CMMI establishes the needed mechanisms
- CMMI and Six Sigma compliment each other
  - CMMI can yield behaviors without benefit
  - Six Sigma improvements based solely on data may miss innovative improvements (assumes a local optimum)
- Training over half the staff has resulted in a change of language and culture
  - Voice of Customer, data-driven decisions, causal analysis, etc.
  - Better to understand and use the tools in everyday work than to adopt the “religion”
Contact Information

Rick Hefner, Ph.D.
Director, Process Management

Northrop Grumman Corporation
One Space Park
Redondo Beach, CA 90278

(310) 812-7290
rick.hefner@ngc.com
Introduction
OMG Systems Modeling Language
(OMG SysML™)
and OOSEM Tutorial

By
Abe Meilich, Ph.D.
abraham.w.meilich@lmco.com

Acknowledged
Original Authors:
Sanford Friedenthal
Alan Moore
Rick Steiner

National Defense Industrial Association
9th Annual Systems Engineering Conference
San Diego, CA
October 23, 2006

Copyright © 2006 by Object Management Group.
Published and used by INCOSE and affiliated societies with permission.
Caveat

• These materials have been modified slightly from the original Tutorial given at INCOSE 2006
  – Softcopy of Full Tutorial available at :

• This material is based on version 1.0 of the SysML specification (ad-06-03-01)
  – Adopted by OMG in May ’06
  – Going through finalization process

• OMG SysML Website
  – http://www.omgsysml.org/
Objectives & Intended Audience

At the end of this tutorial, you should understand the:
- Benefits of model driven approaches to systems engineering
- Types of SysML diagrams and their basic constructs
- Cross-cutting principles for relating elements across diagrams
- Relationship between SysML and other Standards
- Introduction to principles of a OO System Engineering Method

This course is not intended to make you a systems modeler!
You must use the language.

Intended Audience:
- Practicing Systems Engineers interested in system modeling
  - Already familiar with system modeling & tools, or
  - Want to learn about systems modeling
- Software Engineers who want to express systems concepts
- Familiarity with UML is not required, but it will help
Topics

• Motivation & Background
• Diagram Overview
• SysML Modeling as Part of SE Process
• OOSEM – Enhanced Security System Example
• SysML in a Standards Framework
• Transitioning to SysML
• Summary
Background
System Modeling

Integrated System Model Must Address Multiple Aspects of a System
Model Based Systems Engineering

Benefits

• Improved communications
• Assists in managing complex system development
  – Separation of concerns
  – Hierarchical modeling
  – Facilitates impact analysis of requirements and design changes
  – Supports incremental development & evolutionary acquisition
• Improved design quality
  – Reduced errors and ambiguity
  – More complete representation
• Early and on-going verification & validation to reduce risk
• Other life cycle support (e.g., training)
• Enhanced knowledge capture
Modeling at Multiple Levels of the System
What is SysML?

- A graphical modelling language in response to the UML for Systems Engineering RFP developed by the OMG, INCOSE, and AP233
  - a UML Profile that represents a subset of UML 2 with extensions

- Supports the specification, analysis, design, verification, and validation of systems that include hardware, software, data, personnel, procedures, and facilities

- Supports model and data interchange via XMI and the evolving AP233 standard (in-process)
What is SysML (cont.)

• *Is* a visual modeling language that provides
  – Semantics = meaning
  – Notation = representation of meaning

• *Is not* a methodology or a tool
  – SysML is methodology and tool independent
UML/SysML Status

• UML V2.0
  – Updated version of UML that offers significant capability for systems engineering over previous versions
  – Finalized in 2005 (formal/05-07-04)

• UML for Systems Engineering (SE) RFP
  – Established the requirements for a system modeling language
  – Issued by the OMG in March 2003

• SysML
  – Industry Response to the UML for SE RFP
  – Addresses most of the requirements in the RFP
  – Version 1.0 adopted by OMG in May ’06 / In finalization
  – Being implemented by multiple tool vendors
Diagram Overview
Relationship Between SysML and UML

UML 2

SysML

UML reused by SysML (UML4SysML)

SysML extensions to UML (SysML Profile)

UML not required by SysML (UML - UML4SysML)
SysML Diagram Taxonomy

SysML Diagram

Behavior Diagram
- Activity Diagram
- Sequence Diagram
- State Machine Diagram

Requirement Diagram
- Use Case Diagram
- Block Definition Diagram
- Internal Block Diagram

Structure Diagram
- Package Diagram

Symbols:
- Same as UML 2
- Modified from UML 2
- New diagram type
4 Pillars of SysML – ABS Example

1. Structure

2. Behavior

3. Requirements

4. Parametrics
Cross Connecting Model Elements

1. Structure

- ibd [block] Anti-LockController [Internal Block Diagram]
  - satisfies «requirement» Anti-Lock Performance
  - allocatedFrom «objectNode» TractionLoss
  - values DutyCycle: Percentage

- d1:TractionDetector
  - allocatedFrom «activity» DetectLossOf Traction

- m1:BrakeModulator
  - allocatedFrom «activity» Modulate BrakingForce

- c1:modulator Interface
  - «allocate» TractionDetector
  - «allocate» BrakeModulator
  - «allocate» Anti-LockController

- «requirement» Anti-LockPerformance
  - id="337" text="The vehicle shall stop from 60 mph within 150 ft on a clean dry surface."
  - VerifiedBy «interaction» MinimumStoppingDistance
  - SatisfiedBy «block» Anti-LockController

2. Behavior

- act PreventLockup [Swimlane Diagram]
  - «allocate» TractionDetector
  - «allocate» BrakeModulator
  - «allocate» Anti-LockController

- DetectLossOf Traction
  - TractionLoss: Modulate BrakingForce
  - allocatedTo «connector» c1:modulator Interface

- «requirement» Anti-LockPerformance
  - id="102" text="The vehicle shall stop from 60 mph within 150 ft on a clean dry surface."

3. Requirements

- «requirement» StoppingDistance
  - id="102" text="The vehicle shall stop from 60 mph within 150 ft on a clean dry surface."
  - VerifiedBy «interaction» MinimumStoppingDistance
  - SatisfiedBy «block» Anti-LockController

- «requirement» Anti-LockPerformance
  - id="337" text="Braking subsystem shall prevent wheel lockup under all braking conditions."

4. Parametrics

- v.chassis.tire. Friction:

- v.brake.abs.m1. DutyCycle:
  - m: v.brake.abs.m1. BrakingForce:

- v.brake.rotor. DutyCycle:
  - f: F: v.brake.rotor. BrakingForce:

- v.Weight:
  - a: v.Weight. VelocityEquation:
  - [a = dv/dt]

- v.Position:
  - [v = dx/dt]
Structural Diagrams

SysML Diagram

- Behavior Diagram
  - Activity Diagram
  - Sequence Diagram
  - State Machine Diagram
  - Use Case Diagram

- Requirement Diagram
  - Block Definition Diagram
  - Internal Block Diagram
  - Package Diagram

- Structure Diagram
  - Parametric Diagram

Same as UML 2
Modified from UML 2
New diagram type
Package Diagram

• Package diagram is used to organize the model
  – Groups model elements into a name space
  – Often represented in tool browser

• Model can be organized in multiple ways
  – By System hierarchy (e.g., enterprise, system, component)
  – By domain (e.g., requirements, use cases, behavior)
  – Use viewpoints to augment model organization

• Import relationship reduces need for fully qualified name (package1::class1)
Package Diagram
Organizing the Model

pkg SampleModel [by diagram type]
- Use Cases
- Requirements
- Behavior
- Structure
- EngrAnalysis

pkg SampleModel [by level]
- Enterprise
- System
- Logical Design
- Allocated Design
- Verification

pkg SampleModel [by IPT]
- Architecture Team
- Requirements Team
- IPT A
- IPT B
- IPT C

By Diagram Type
By Hierarchy
By IPT
Package Diagram - Views

- Model is organized in one hierarchy
- Viewpoints can provide insight into the model using another principle
  - E.g., analysis view that spans multiple levels of hierarchy
  - Can specify diagram usages, constraints, and filtering rules
  - Consistent with IEEE 1471 definitions
Blocks are Basic Structural Elements

• Provides a unifying concept to describe the structure of an element or system
  – Hardware
  – Software
  – Data
  – Procedure
  – Facility
  – Person

• Multiple compartments can describe the block characteristics
  – Properties (parts, references, values)
  – Operations
  – Constraints
  – Allocations to the block (e.g. activities)
  – Requirements the block satisfies

```
«block»

BrakeModulator

allocatedFrom

«activity»Modulate

BrakingForce

values

DutyCycle: Percentage
```
Block Property Types

- Property is a structural feature of a block
  - **Part property** aka. part (typed by a block)
    - Usage of a block in the context of the enclosing block
    - Example - right-front:wheel
  - **Reference property** (typed by a block)
    - A part that is not owned by the enclosing block (not composition)
    - Example - logical interface between 2 parts
  - **Value property** (typed by value type)
    - Defines a value with units, dimensions, and probability distribution
    - Example
      - Non-distributed value: tirePressure:psi=30
      - Distributed value: «uniform» {min=28,max=32} tirePressure:psi
Using Blocks

- Based on UML Class from UML Composite Structure
  - Eliminates association classes, etc.
  - Differentiates value properties from part properties, add nested connector ends, etc.
- Block definition diagram describes the relationship among blocks (e.g., composition, association, classification)
- Internal block diagram describes the internal structure of a block in terms of its properties and connectors
- Behavior can be allocated to blocks
Block Definition vs. Usage

**Definition**
- Block is a definition/type
- Captures properties, etc.
- Reused in multiple contexts

**Usage**
- Part is the usage in a particular context
- Typed by a block
- Also known as a role
Internal Block Diagram (ibd)
Blocks, Parts, Ports, Connectors & Flows

Internal Block Diagram Specifies Interconnection of Parts
Reference Property Explained

S1 is a reference part in ibd shown in dashed outline box

ibd [block] Anti-LockController [Internal Block Diagram]

c2: sensor Interface

s1: Sensor

c1: modulator Interface

d1: Traction Detector

m1: Brake Modulator

bdd [package] VehicleStructure
SysML Port

- Specifies interaction points on blocks and parts
  - Supports integration of behavior and structure

- Port types
  - Standard (UML) Port
    - Specifies a set of operations and/or signals
    - Typed by a UML interface
  - Flow Port
    - Specifies what can flow in or out of block/part
    - Typed by a flow specification

2 Port Types Support Different Interface Concepts
Port Notation

Standard Port

provided interface
(provides the operations)

Flow Port

required interface
(calls the operations)

part1:

item flow

part2:
Parametrics

- Used to express constraints (equations) between value properties
  - Provides support for engineering analysis (e.g., performance, reliability)
- Constraint block captures equations
  - Expression language can be formal (e.g., MathML, OCL) or informal
  - Computational engine is defined by applicable analysis tool and not by SysML
- Parametric diagram represents the usage of the constraints in an analysis context
  - Binding of constraint usage to value properties of blocks (e.g., vehicle mass bound to \( F = m \times a \))

Parametrics Enable Integration of Engineering Analysis with Design Models

11 July 2006 Copyright © 2006 by Object Management Group.
Defining Vehicle Dynamics

Defining Reusable Equations for Parametrics
Vehicle Dynamics Analysis

Using the Equations in a Parametric Diagram to Constrain Value Properties
Behavioral Diagrams

SysML Diagram

Behavior Diagram

Activity Diagram

Sequence Diagram

State Machine Diagram

Use Case Diagram

Use Case Diagram

Block Definition Diagram

Internal Block Diagram

Package Diagram

Parametric Diagram

Same as UML 2

Modified from UML 2

New diagram type
Activities

• Activity used to specify the flow of inputs/outputs and control, including sequence and conditions for coordinating activities
• Secondary constructs show responsibilities for the activities using swim lanes
• SysML extensions to Activities
  – Support for continuous flow modeling
  – Alignment of activities with Enhanced Functional Flow Block Diagram (EFFBD)
Activity Diagram Notation

- Join and Merge symbols not included
- Activity Parameter Nodes on frame boundary correspond to activity parameters
Activity Diagrams
Pin vs. Object Node Notation

• Pins are kinds of Object Nodes
  – Used to specify inputs and outputs of actions
  – Typed by a block or value type
  – Object flows connect object nodes

• Object flows between pins have two diagrammatic forms
  – Pins shown with object flow between them
  – Pins elided and object node shown with flow arrows in and out

Pins must have same characteristics (name, type etc.)
Explicit Allocation of Behavior to Structure Using Swimlanes

Activity Diagram (without Swimlanes)

Activity Diagram (with Swimlanes)
SysML EFFBD Profile

EFFBD - Enhanced Functional Flow Block Diagram

Aligning SysML with Classical Systems Engineering Techniques
Distill Water Activity Diagram (Continuous Flow Modeling)

Representing Distiller Example in SysML Using Continuous Flow Modeling
Activity Decomposition

**Definition**

**Use**

11 July 2006

Copyright © 2006 by Object Management Group.
Interactions

• Sequence diagrams provide representations of message based behavior
  – represent flow of control
  – describe interactions

• Sequence diagrams provide mechanisms for representing complex scenarios
  – reference sequences
  – control logic
  – lifeline decomposition

• SysML does not include timing, interaction overview, and communications diagram
Black Box Interaction (Drive)

UML 2 Sequence Diagram Scales
by Supporting Control Logic and Reference Sequences
Black Box Sequence (StartVehicle)

Simple Black Box Interaction

References Lifeline Decomposition For White Box Interaction

driver:Driver
State Machines

- Typically used to represent the life cycle of a block
- Support event-based behavior (generally asynchronous)
  - Transition with trigger, guard, action
  - State with entry, exit, and do-activity
  - Can include nested sequential or concurrent states
  - Can send/receive signals to communicate between blocks during state transitions, etc.
Operational States (Drive)

Transition notation: trigger[guard]/action
Use Cases

• Provide means for describing basic functionality in terms of usages/goals of the system by actors
• Common functionality can be factored out via include and extend relationships
• Generally elaborated via other behavioral representations to describe detailed scenarios
• No change to UML
Operational Use Cases

Cases [Operation]
Cross-cutting Constructs

• Allocations
• Requirements
Allocations

• Represent general relationships that map one model element to another

• Different types of allocation are:
  – Behavioral (i.e., function to component)
  – Structural (i.e., logical to physical)
  – Software to Hardware
  – ....

• Explicit allocation of activities to structure via swim lanes (i.e., activity partitions)

• Both graphical and tabular representations are specified
Different Allocation Representations (Tabular Representation Not Shown)

Allocate Relationship

Explicit Allocation of Activity to Swim Lane

Compartment Notation

Callout Notation
SysML Allocation of SW to HW

- In UML the deployment diagram is used to deploy artifacts to nodes
- In SysML allocation on ibd and bdd is used to deploy software/data to hardware
Requirements

• The «requirement» stereotype represents a text based requirement
  – Includes id and text properties
  – Can add user defined properties such as verification method
  – Can add user defined requirements categories
    (e.g., functional, interface, performance)
• Requirements hierarchy describes requirements contained in a specification
• Requirements relationships include DeriveReqt, Satisfy, Verify, Refine, Trace, Copy
Requirements Breakdown

```
req [package] HSUVRequirements [HSUV Specification]

HSUVSpecification

«requirement» Eco-Friendliness

«requirement» Performance

«requirement» Braking

«requirement» FuelEconomy

«requirement» Acceleration

RefinedBy
«useCase» HSUVUseCases::Accelerate

«requirement» Emissions

Id = “R1.2.1”
text = “The vehicle shall meet Ultra-Low Emissions Vehicle standards.”

«deriveReqt»

SatisfiedBy
«block» PowerSubsystem

VerifiedBy
«testCase» MaxAcceleration
```

Requirement Relationships Model the Content of a Specification
Example of Derive/Satisfy Requirement Dependencies

Client depends on supplier (i.e., a change in supplier results in a change in client)

Arrow Direction Opposite Typical Requirements Flow-Down
Problem and Rationale can be attached to any Model Element to Capture Issues and Decisions.
SysML Modeling
as Part of the SE Process
OOSEM – Enhance Security System (ESS) Example
System Development Process

Integrated Product Development (IPD) is essential to improve communications.

A Recursive V process that can be applied to multiple levels of the system hierarchy.
Common Subactivities

- Mission use cases/scenarios
- Enterprise model
- System use cases/scenarios
- Elaborated context
- Req’ts diagram
- Logical architecture
- Node diagram
- HW, SW, Data architecture

Define System Requirements

Define Logical Architecture

Validate & Verify System

Optimize & Evaluate Alternatives

Engr Analysis Models
- Trade studies

Test cases/procedures

Analyze Needs
Enhanced Security System Example

- The Enhanced Security System is the example for the OOSEM material
  - Problem fragments used to demonstrate principles
  - Utilizes Artisan RTS™ Tool for the SysML artifacts
ESS Requirements Flowdown

- **Market Needs**
  - **Intruder Detection**
    - **System Specification**
      - **Logical Requirements**
        - **Allocated Requirements**

- **Requirement**
  - **SS102**
    - System shall detect intruder entry and exit ...

- **Verification**
  - Entry/Exit Subsystem
  - Entry/Exit Detection Test

- **Trace**
  - Satisfied by Entry/Exit Subsystem
  - Verified by Entry/Exit Detection Test

ESS Operational Enterprise To-Be Model

Domain To-Be

«enterprise»
ESS Operational Enterprise

«moe» OperationalAvailability = {>.99}
«moe» MissionResponseTime = {<5 min}
«moe» OperationalCost = {TBD}
«moe» CostEffectiveness

MonitorSite ()
DispatchEmergencyServices ()
ProvideEmergencyResponse ()

Protected Site

«external»
Physical Environment

«external»
Single-family Residence

«external»
Multi-family Residence

«external»
Business

«system»
ESS

«external»
Comm Network

«external»
Emergency Services

Assess Report ()
Report Update ()
Dispatch Police ()

Dispatcher
Responder

Fire
Paramedic

Police

«package» ESS Enterprise (To Be)

Intruder

1..*

Customer

1..*
System Use Cases - Operate

- Activate/Deactivate
- Monitor Site
- Respond to Break-In
- Respond to Fire
- Respond to Medical
System Scenario: Activity Diagram

Monitor Site (Break-In)
ESS Logical Design – Example Subsystem
ESS Allocation Table (partial)

- Allocating Logical Components to HW, SW, Data, and Procedures components

<table>
<thead>
<tr>
<th>Logical Components</th>
<th>Entry Sensor</th>
<th>Exit Sensor</th>
<th>Perimeter Sensor</th>
<th>Entry/Exit Monitor</th>
<th>Event Monitor</th>
<th>Site Comms I/F</th>
<th>Event Log</th>
<th>Customer I/F</th>
<th>Customer Output Mgr</th>
<th>System Status</th>
<th>Fault Mgr</th>
<th>Alarm Generator</th>
<th>Alarm I/F</th>
</tr>
</thead>
<tbody>
<tr>
<td>«software»</td>
<td>Device Mgr</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>SF Comm I/F</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>User I/F</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Event Mgr</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Site Status Mgr</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Site RDBMS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>CMS RDBMS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>«data»</td>
<td>Video File</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>CMS Database</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Site Database</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>«hardware»</td>
<td>Optical Sensor</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>DSL Modem</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>User Console</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Video Camera</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Alarm</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>
ESS Parametric Diagram
To Support Trade-off Analysis

\[
\text{CostEffectiveness} = \text{Sum}(w_1 \cdot u(\text{OA}) + w_2 \cdot u(\text{MRT}) + w_3 \cdot u(\text{OC}))
\]
Entry/Exit Test Case

**Description**

- Intruder enters through front door
- Door sensor detects entry
- New alert status sent to central system
- Intruder leaves through lounge window
- Window sensor detects exit
- Changed alert status sent to central system

**Diagram:**

- **Intruder Emulator (testComponent):**
  - Door[1]: Optical Sensor
  - Window[4]: Optical Sensor
  - Site Processor
  - DSL Modem

- **Sequence Diagram:**
  - Intruder enters through front door
  - Door sensor detects entry
  - SensedEntry
  - New alert status sent to central system
  - Intruder leaves through lounge window
  - Window sensor detects exit
  - SensedExit
  - Changed alert status sent to central system
SysML in a Standards Framework
Systems Engineering Standards Framework (Partial List)

Process Standards
- EIA 632
- ISO 15288
- IEEE 1220
- CMMI

Architecture Frameworks
- FEAF
- DoDAF
- MODAF
- Zachman FW

Modeling Methods
- HP
- OOSE
- SADT
- Other

Modeling & Simulation Standards
- IDEF0
- SysML
- UPDM
- System Modeling

Interchange & Metamodelling Standards
- HLA
- MathML
- Simulation & Analysis

Implementation By Tools
- MOF
- XMI
- STEP/AP233
- Data Repository
ISO/IEC 15288
System Life Cycle Processes

Enterprise Processes

5.3.2 Enterprise Environment Management Process
5.3.3 Investment Management Process
5.3.4 System Life Cycle Processes Management
5.3.5 Quality Management Process
5.3.6 Resource Management Process

Agreement Processes

5.2.2 Acquisition Process
5.2.3 Supply Process

Project Processes

5.4.2 Project Planning Process
5.4.3 Project Assessment Process
5.4.4 Project Control Process
5.4.5 Decision-Making Process
5.4.6 Risk Management Process
5.4.7 Configuration Management Process
5.4.8 Information Management Process

Technical Processes

5.5.2 Stakeholder Reqs Definition Process
5.5.3 Reqs Analysis Process
5.5.4 Architectural Design Process
5.5.5 Implementation Process
5.5.6 Integration Process
5.5.7 Verification Process
5.5.8 Transition Process
5.5.9 Validation Process
5.5.10 Operation Process
5.5.11 Maintenance Process
5.5.12 Disposal Process

11 July 2006 Copyright © 2006 by Object Management Group.
Standards-based Tool Integration with SysML

Systems Modeling Tool

Other SE Engineering Tools

Model/Data Interchange

AP233/XMI

AP233/XMI

11 July 2006

Copyright © 2006 by Object Management Group.
Participating SysML Tool Vendors

- Artisan
- EmbeddedPlus
  - 3rd party IBM vendor
- Sparx Systems
- Telelogic (includes I-Logix)
- Vitech

Note: Free Visio SysML Template available at OMG SysML site (http://www.omgsysml.org)
UML Profile for DoDAF/MODAF (UPDM) Standardization

• Current initiative underway to develop standard profile for representing DODAF and MODAF products
  – Requirements for profile issued Sept 05
  – Final submissions expected Dec ‘06
• Multiple vendors and users participating
• Should leverage SysML
Transitioning to SysML
Using Process Improvement
To Transition to SysML

Continuous Improvement Cycle

Plan Improvement

Define Improvement

Pilot Improvement

Deploy Improvement

Assess & Measure Improvement
Integrated Tool Environment

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>System Modeling</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SysML</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Software Modeling</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>UML 2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Hardware Modeling</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>VHDL, CAD, ..</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Verification &amp; Validation</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Specialty Engineering Analysis</td>
</tr>
</tbody>
</table>
Summary and Wrap up
Summary

• SysML sponsored by INCOSE/OMG with broad industry and vendor participation
• SysML provides a general purpose modeling language to support specification, analysis, design and verification of complex systems
  – Subset of UML 2 with extensions
  – 4 Pillars of SysML include modeling of requirements, behavior, structure, and parametrics
• OMG SysML Adopted in May 2006
• Multiple vendor implementations announced
• Standards based modeling approach for SE expected to improve communications, tool interoperability, and design quality
References

- OMG SysML website
  - http://www.omgsysml.org
- UML for Systems Engineering RFP
  - OMG doc# ad/03-03-41
- UML 2 Superstructure
  - OMG doc# formal/05-07-04
- UML 2 Infrastructure
  - OMG doc# ptc/04-10-14
Systems Engineering
for the
Joint Capabilities Integration and Development System
(JCIDS)

Tutorial for the 9th NDIA Systems Engineering Conference
Agenda and Presenters

- Introduction to JCIDS – Chris Ryder
- Applying the Systems Engineering Method for JCIDS – Dave Flanigan
- Model-Driven Systems Engineering for JCIDS – Jennifer Rainey
- JCIDS Functional Analyses – Dave Krueger and Chris Ryder
- Summary – Chris Ryder
Purpose of the Tutorial

- JCIDS prescribes a joint forces approach to identify capability gaps against current force capability needs
- The Systems Engineering (SE) Method applies to each iteration of the systems life-cycle from capability inception through system retirement
- Good systems engineering practice is necessary for successfully implementing JCIDS
- JCIDS Functional Analyses perform critical problem solving activities
- Use of model-driven SE facilitates JCIDS throughout the systems life-cycle
What is JCIDS?

- Capabilities-based assessment (CBA) composed of a structured
- Four-step methodology that defines capability gaps
- Capability needs and approaches to provide those capabilities within a specified functional or operational area.
JCIDS Is an Engineering Intensive Function

- JCIDS activities are fundamental Systems Engineering actions
  - Consistent with the Systems Engineering Method
  - Performed at early concept analysis and development
  - But also at each capability upgrade
- JCIDS analysis quantifies material and non-material options
  - Systems Engineering life-cycle phases quantifies the phases of “Materialization”
    - Abstract concepts in early phases
    - Concrete systems and subsystems as the life cycle progresses
JCIDS Process

DOD Strategic Guidance

Joint Operations Concepts

Joint Operating Concepts
Joint Functional Concepts
Joint Integrating Concepts

Functional Area Analysis

Functional Needs Analysis

JCD

DOTLPF Analysis (Non-materiel Approaches)

Ideas for Materiel Approaches

Analysis of Materiel/Non-materiel Approaches

Alternative N
Alternative 2
Alternative 1

Integrated Architectures

Post Independent Analysis

CPD
CDD
ICD
DCR

*DOD 3170.01B*
JCIDS Events

- **Functional Area Analysis (FAA)**
  - Identify operational task, conditions, and standards needed to accomplish military objectives
  - **Result:** Tasks to be accomplished

- **Functional Needs Analysis (FNA)**
  - Assess ability of current and programmed capabilities to accomplish the tasks
  - **Result:** List of capability gaps

- **Functional Solutions Analysis (FSA)**
  - Operational based assessment of DOTMLPF approaches to solving capability gaps
  - **Result:** Potential DOTMLPF approaches to capability gaps

- **Post Independent Analysis**
  - Independent analysis of approaches to determine best fit
  - **Result:** Initial Capabilities Document
JCIDS analytical process stresses the fundamentals for applying an effective systems engineering program by any accepted standard.

It guides the “front-end” phases of the SE process for each capability iteration:
- Enterprise (operational) analysis
- Requirements definition
- Life-cycle phase

The analysts must have a thorough understanding of existing capabilities as well as the capability needs.

The JCIDS analysis team eventually determines the optimum combination of material and non-material alternatives to achieve the capability needs to the Battle Force.
Perspective

- Not an authoritative review of DoD policy and procedures
  - Chairman of the Joint Chiefs of Staff Instruction (CJCSI) 3170.01, “Joint Capabilities Integration And Development System”
  - Chairman of the Joint Chiefs of Staff Manual (CJCSM) 3170.01, “Operation of the Joint Capabilities Integration and Development System”

- Relationship to the acquisition process

- Discuss some thoughts on implementation
  - FAA, FNA, FSA, and PIA
  - No definitive cookbook for implementation
The JCIDS Meta-Model

- OMG SysML™ model of JCIDS activities and artifacts
  - Activities
    - FAA
    - FNA
    - FSA
    - PIA
  - Artifacts
    - Architecture model
      - Including capability use cases
    - ICD, CDD, CPD
    - DOTMLPF Change Recommendations
    - Integrated Threat Warning Assessment (ITWA)
    - CONOPS
OMG SysML™

- Diagrams used throughout this presentation are constructed using the Object Management Group’s SysML (Systems Modeling Language)
- Systems engineering extension to Unified Modeling Language (UML™) 2.0
- OMG SysML is a standardized family of diagrams depicting system elements, their behaviors and their relationships with other elements internal and external to the system
  - Captures operational and systems requirements
  - Documents element parametrics and constraints
  - Methodology independent

For a detailed discussion on SysML, each of you are invited to attend Abe Meilich’s SysML Tutorial this afternoon
Targeting and Bomb Damage Assessment

- Read the attached “Statement of Operational Need” at the break
- TBDA represents a required capability to:
  - Maintain persistent coverage over a target area
  - Acquire fire-control quality track files on moving targets
  - Provide the means to determine whether a target was sufficiently destroyed or neutralized
  - Must be able to be deployed by a small team of ground-based personnel
  - Controlled by the local ground commander

TBDA is a FICTICIOUS Case! Any similarity with any capability or system, real or imagined, is purely coincidental!
This case presents a very limited sample of artifacts and elements that are part of the FAA, FNA and FSA SE Model.

Intent is to illustrate modeling possibilities using SysML:
- Requirements traceability
- Entities with their behaviors and relationships
  - Material and non-material
- Standards that govern architectural elements
Systems Engineering Method

David Flanigan

October 23, 2006
Discussion Topics

- Describe Systems Engineering Method (SEM) in JCIDS context
  - Identify and describe four “root” steps
  - Identify inputs/conditions for each step
  - Identify outputs/products from each step
  - Inter-relationships among the steps
- Show linkage to JCIDS and the Systems Engineering lifecycle
Regardless of the analytical phase performed by the JCIDS SE team,

- The basic application of the SE method is constant throughout the process

Each SE Method activity is performed in some form or to some degree in each phase of the system life-cycle
Systems Engineering Method

Adapted from Kossiakoff & Sweet “Systems Engineering Principles and Practice”

Crossing the boundary from the problem space to the solution space costs $$
The Systems Engineering “V” is iterative and tailored throughout the entire lifecycle.
The generic SE method can be mapped to each specific iteration of the SE “V”
Systems Engineering Method Over Life Cycle

The front end of JCIDS has no Systems Engineering “V”
Systems Engineering Method Over Life Cycle

JCIDS is performed over the entire lifecycle

Need to do good SE at the start of the lifecycle, since each phase relies on products from the previous phase
Ensure the traceability of needs and requirements are present through the entire lifecycle
Systems Engineering Method

Problem Definition

Legacy Operational Activities

Capabilities

Rationale, Scope, & Context

Sponsor-derived Problem Set

Functional Decomposition

Technological Contributions

Functional Improvements

Directed Functions

Material / Non-Material solution?

Material

Non-Material

DOTMLPF Elements

Collect Candidate Systems

Requirements (Capability) Analysis

Analysis

Problem satisfied?

Yes

No

Design Validation

Physical Definition
Systems Engineering Method
Phase 1: Requirements (Capability) Analysis

Requirements (Capability) Analysis

Problem Definition

Legacy Operational Activities

Rationale, Scope, & Context

Capabilities

Sponsor-derived Problem Set

To Functional Analysis Phase
Problem Definition

- At one point in time there is a problem that must be solved due to:
  - Deficient capability with existing systems
  - Desire to improve existing performance
- Need to understand what the objectives are to provide the desired capability
- Define the operational context within the Capability Enterprise!

Did we define the problem correctly?
Did we define the correct problem???
Example Requirements Analysis Products

- Clear(er) definition of the problem
- Proper scope of the problem
- Operational context documents and data bases
  - Design Reference Mission
  - Strategy-to-Task Mapping
  - Concept of Operations
  - Physical Environment Database
  - Threat Representation Database
  - Blue Capabilities Database
- Relevant Operational Views

Captured within a SE Requirements Model
Systems Engineering Method
Phase 2: Functional Definition

From Requirements Definition

- Functional Improvements
- Directed Functions
- Functional Decomposition
- Technological Contributions

Functional Definition

To Physical Definition
Typical Functional Definition Products

- Functional Decomposition of required activities
  - Functional diagrams (Functional Flow Block Diagrams, UML Activity Diagrams)
- Associated metrics with these functions (threshold / objective)
- Analysis process that determines if you can solve with a material / non-material / both solution
  - Be able to document and defend this process
- How do we know it’s right?
  - The functions are legitimate, correct, and validated by users
- Functional Area Analysis
- Relevant operational views

Functional Analysis Documented in a SE Functional or Physical Model
Systems Engineering Method
Phase 3: Physical Definition

From Functional Definition

Non-Material

Material / Non-Material solution?

Material

DOTMLPF Elements

Collect Candidate Systems

Physical Definition

To Design Validation
Typical Physical Definition Products

- Provide system alternatives towards satisfying required functionality
  - Assignment of functions to physical elements
- DOTMLPF analysis products
  - Based on the functional definition phase
- CONOPS changes / recommendations
  - Based on DOTMLPF analysis
- Risk management strategies of the system
- System roadmaps to bridge the gap between the current and future capabilities
- Functional Needs Analysis
- Relevant operational and SYSTEMS views

SE Physical Model with Physical Definition Begins
Evolution Toward a Systems Model
Systems Engineering Method
Phase 4: Design Validation

From Physical Definition

1. Analysis
2. M&S
3. Problem satisfied?
   - Yes: Design Validation
   - No: Reassess requirements, functional elements or physical details
5. To next life-cycle phase: Requirements Definition
Typical Design Validation Products

- Demonstrate the analysis documents the assumptions, follows a rigorous process, and arrives at meaningful conclusions that are justifiable
  - There may be multiple processes and products dependent on the sponsor, personnel/time availability, experience
  - This may be an iterative process for ICD, CDD, CPD
- Trade studies
- VV&A
- Risk Management
- Cost Analysis
- Force Allocation
- Functional Solutions Analysis
- Program Independent Assessment

Attain a Fully Validated Systems Engineering Model
Systems Engineering Methodology  
Linkage to JCIDS Summary

- SE methods can be used to produce JCIDS products/artifacts
- SE methods can iterate throughout the DoD 5000 lifecycle
- Good SE methods can produce JCIDS
- Bad SE methods can produce JCIDS
- Producing JCIDS does not guarantee good SE

Good SE ↔ Effective JCIDS
Applying Model-Driven Systems Engineering Practices to JCIDS

Jennifer Rainey

October 23, 2006
Discussion Topics

- Purpose – a model-driven systems engineering (MDSE) approach supports the JCIDS lifecycle process

- What is Model-Driven Systems Engineering?

- How to Apply Model-driven Systems Engineering for JCIDS?
Purpose

- A Model Driven Systems Engineering (MDSE) approach supports the entire defense acquisition, technology, and logistics lifecycle
  - A systems engineering model provides traceability from system development back to initial JCIDS process and war fighting need
  - MDSE focuses on techniques that drive capability identification
    - Documents entire system lifecycle
    - Identifies the capabilities, capability gaps, and materiel/non-materiel solutions
    - Develops foundation for integrated architectures
  - JCIDS is a concept-centric capabilities identification process
    - “The process to identify capability gaps & potential material and non-materiel solutions must be supported by a robust analytical process that incorporates innovative practices…”
- CJCSI 3170.01E 11 May 2005

Use of model-driven SE facilitates JCIDS throughout the systems lifecycle
Integrated Defense Acquisition, Technology, & Logistics Lifecycle Management Framework

MDSE should be used here

MDSE is used here
Systems Engineering Model

- A model is a representation of a system
  - Assists stakeholders, including engineers, to understand something that is not easily comprehensible
  - Communicates the organization of the system to stakeholders
  - Enhances understanding of interfaces, relationships, operations and risk
  - Continually updated

- Systems Engineering Model
  - Build as the basis for JCIDS analysis
  - Covers the problem and solution space
  - Contains the objects, relationships and the data
    - Requirements, Functional, and Physical
  - Develops the integrated architecture

Systems Engineering Model is a Living Entity
Model-Driven Systems Engineering

- Establish system model bases on:
  - Requirements model
  - Functional model
  - Physical model

- Show relationships between the models
  - Link operational needs to capabilities
  - Link capabilities to requirements
  - Link requirements to functions
  - Link functions to systems

“If you don’t model it, you won’t understand it.”

Ivar Jacobson
Systems Engineering Method

Building blocks of an integrated architecture

Adapted from Kossiakoff & Sweet “Systems Engineering Principles and Practice”
Requirements Model

- Requirements Analysis
  - Define/scope the problem space
  - Identify “capabilities” during JCIDS process to meet war fighting needs
    - Capabilities turn into requirements later in the lifecycle
  - Analyze capabilities/requirements
    - Assess against “as-is” capabilities/systems, identify gaps
    - Ensure they are necessary, concise, attainable, complete, consistent, unambiguous, and verifiable
    - Create requirements traceability
  - Products:
    - Framework for Operational Views
      - Fulfill need to develop DoDAF operational view artifacts
      - Sets standards to be used needed for technical view artifacts
    - Metrics
      - Measures of Effectiveness
      - Measures of Performance
    - Operational context documents
Functional Model

- Functional Definition
  - Implementation free identification of required activities
  - Establish functional decomposition
    - Use Cases, Operational Scenarios
    - Functional Flow Block Diagrams (FFBDs)
    - Unified Modeling Language (UML) Activity Diagrams
  - Can model the time sequencing of the functions
  - Show data or information flow between functions
    - Fulfill need to develop DoDAF operational view artifacts
    - Fulfill need to develop DoDAF system view artifacts
  - Products:
    - Capabilities/functionality needed to meet requirements
    - Refined performance metrics
    - Framework for Operational and System Views
Physical Model

- **Physical Definition** – solution space
  - Set the system context or boundary
    - Context diagrams
    - Class diagrams

- Allocate functions to physical elements
  - Evaluate “to-be” capabilities against “as-is” capabilities and systems to identify the “capability gaps” and “redundancies”
  - Establish link to requirements

- **Products**:
  - System Elements
  - Relevant system views
  - System data exchanges
  - System roadmaps to bridge the capability gap

**Formed the Systems Engineering Model**
The Systems Engineering Model

- Where it all ties together!
  - Formed by establishing the relationships between the requirements, functional, and physical elements of the model
    - Requirements (capabilities) link to functions, functions are allocated to physical components
      - Early in the process, the system solution can be expressed as a “black box”
      - As the lifecycle advances, the physical model is further refined into sub-systems
        - Ensure every requirement is linked to a function
        - Ensure every function is allocated to physical element
    - The MDSE process forms the basis for the integrated architecture
  - Supports impact analysis
    - The SE Model developed during upfront JCIDS is the same model used during the entire acquisition lifecycle
    - Traceability is maintained back to the original capability need identified
      - Allows greater understanding of the impact of how changing one element in the model, impacts other areas
MDSE Basis for Integrated Architectures

Integrated Defense Acquisitions, Technology, & Logistics Life Cycle Management Framework

Functional Area Analysis

Functional Needs Analysis

DOTLPF Analysis (Non-materiel Approaches)

Ideas for Materiel Approaches

Analysis of Materiel/Non-materiel Approaches

Alternative N

Alternative 2

Alternative 1

Post Independent Analysis

Integrated Architectures

Joint Operating Concepts

Joint Functional Concepts

Joint Integrating Concepts

DOD Strategic Guidance

CPD

CDD

ICD

DCR

Functional Solution Analysis

DOD Strategic Guidance

JCD

Integrated Defense Acquisitions, Technology, & Logistics Life Cycle Management Framework

Joint Operations Concepts
Architectures in JCIDS

- “Integrated Architectures” are a foundation for the analytical process
  - Stated requirements, attributes, and measures
  - Meets DoDAF needs
  - Used during upfront JCIDS, concept refinement, technology development, system integration, system development, and production
  - System model defines the architecture used during the lifecycle

- “Key components of the CDD and CPD are the integrated architecture products that ensure the DoD understands the linkages between capabilities and systems and can make appropriate acquisition decisions.” CJCSI 3170.01E 11 May 2005

The “Systems Model” becomes the basis for architecture and JCIDS analysis
Architecture Views

- Architecture Views
  - A view is a different “slice” of the model
  - Provides a look “inside” the model
  - Includes information relevant to the stakeholder

- An architecture engenders a multitude of artifacts
  - Most are derived using the same information and data elements
    - Can be obtained from the systems engineering model
  - DoDAF architecture views are specific types of artifacts
    - Includes Operational, Systems, and Technical Views
  - DoDAF architecture views are just a few of the possible model views
JCIDS Systems Engineering Model

- A Systems Engineering model captures the essential elements of the systems engineering life-cycle

- “Dynamic and recursive process” (Bootch, Rumbaugh, Jacobson)
  - Iteratively captures enterprise capabilities and system requirements
  - Promotes incorporation of technology evolution

- Forms basis for sound, long-term systems engineering and analysis
  - Compliant with DoDAF and JCIDS

Model-Driven SE in Defense Systems Acquisition becomes Model-Driven JCIDS
How to Apply MDSE to JCIDS

- Establish a meta-model to understand the framework for the process

- Meta-model is another abstraction, highlighting the properties of a model
  - *Explicit* description (constructs and rules) of how a domain-specific model is built

- JCIDS meta-model is composed of:
  - Dynamic elements – modeling the behavior over time
  - Logical elements – static view of the objects and classes

Need to model JCIDS process as a “meta-model”
The JCIDS Meta-Model

- **Dynamic Component**
  - Incorporates model-driven analyses within the JCIDS process
  - Standardizes SE modeling methods demonstrate utility for modeling JCIDS capabilities
  - Applies the model-driven approach to each JCIDS analytical phase
    - Leading up to JCIDS analyses documentation
    - Appropriate for capability iterations throughout the Warfare Systems’ lives
  - Easily updated and maintained
  - Use throughout the acquisition lifecycle

- **Logical Component**
  - The Capability Object exists within the “Capability Enterprise”
  - Captures logical and dynamic elements
  - Identifies the attributes and operations of a Capability Object functioning within the operational domain
  - Identifies “Non-Materiel” elements of DOTMLPF
JCIDS Meta-Model Dynamic Component

- FAA, FNA, FSA, and PIA are represented as use cases
  - Each phase represents a dynamic set of activities
  - With post-condition “Result of Value”

- Relates the JCIDS activities to the process of SE/Architecture modeling
  - Understanding the As-Is Enterprise and evolving the To-Be mission scenarios and use cases
JCIDS Dynamic Model

Performance Object

- Perform FAA
- Perform FNA
- Perform FSA
- Conduct PIA

Capability Sponsor

<<Capability>>
Capability Object
JCIDS Meta-Model Logical

- Focus of analysis is on a Capability Object
  - Enables itself within the Capability Enterprise
- Identification of needed capabilities to fulfill war fighting needs (FAA)
- Baseline Capability Enterprise is composed of As-Is capabilities of legacy As-Is Warfare System
- Comparison of To-Be capabilities against the As-Is baseline yields the Capability Gap(s) (FNA)
- Evolve the capability and allocate to physical To-Be Warfare System (FSA)
- DOTMLPF applies needs analysis and potential solutions
Capability Object

- Form of “System Object” as defined by Object Oriented Systems Engineering Methods (OOSEM)
  - Performs operations on behalf of itself and/or other objects
    - Provide output result of value
    - Provide services and information related elements within the domain
  - Possesses measurable properties
    - Physical, data, performance

- Capability Objects, like all UML classes, possess:
  - Attributes
  - Operations
  - Associations
Capability Object and the Warfare System

class Capabilities Enterprise

- <<capability>> Capability Enterprise
- Component of
- + Supports/Affects
- + Affected/Enhanced by
- Assigned to
- + Attributes
- + Tasks()

- <<system>> Warfare System
- <<system>> System As-Is
- <<system>> System To-Be
- <<capability>> Capability Object
- <<capability>> Capability As-Is
- <<capability>> Capability To-Be
- DOTMLPF
- DOTMLPF As-Is
- DOTMLPF To-Be
Capability Object

- **Package**: Capability Object
- **Attributes**:
  - `Task()`
- **Class Association**
- **Scenario**
- **Operations**
  - `Employed In`
  - `Uses`
- **Attributes**:
  - `Property Type`
  - `Parameter`
  - `Evolution`
  - `Function`
- **Measurements**
  - `Effectiveness`
  - `Performance`
  - `Suitability`

**Institutions**
- FAA
- FNA/FSA
- FSA
- ICD/CDD/CPD
Transition from Capability to System

- Use capability object to perform assessments to satisfy DOTMLPF
  - Analysis of Materiel/non-materiel approaches
  - Analysis of Alternatives
  - Initial Capability Document
  - Investigate if a modification to any element of DOTMLPF except the “M” will enhance the Capability Enterprise
    - A far less expensive option
  - DOTMLPF elements can be modeled as classes
    - Each non-materiel element possess attributes and operations
    - Helpful to define a meta-class early in the process to understand element components and relationship
Transition from Capability to System

The Legacy System

DOTMLPF can also transition from “As-Is” to “To-Be”
Model-driven Approach Facilitates JCIDS

Building blocks of an integrated architecture

Adapted from Kossiakoff & Sweet "Systems Engineering Principles and Practice"
Benefits of Model-Driven Approach

- Traceable back to initial FAA and war fighting need
  - Changes to system requirements can be evaluated against the “to-be” capability identified during FAA, FNA, and FSA
    - Ensures solution implemented meets intent of JCIDS analysis

- One place to document entire system lifecycle from inception to deployment
  - Document rationale for decisions and analysis
  - Easily supports changes/updates to the model while maintaining historical information
  - Abstracts the complexities of the warfare system, the capability system, and associated elements such that a team can effectively grasp them

- Appropriate integrated architecture views can be generated
  - Operational views – requirements and functional model
  - System views – physical model
  - Technical views – requirements, physical models
Summary

- Model-driven SE will provide robust lifecycle system model
  - Provides integrated architecture
  - Supports initial capabilities assessment
  - Establishes framework for entire lifecycle: concept refinement, technology development, system development and demonstration, production and deployment, and operations and support phases

- Systems Engineering methodology enhances the JCIDS process
  - Models abstract complexities of modern warfare systems

- Comprehensive models provide for compilation of data needed to assess capabilities and comply with JCIDS

Models bridge the diverse knowledge domains of the warrior and the engineer
Functional Area Analysis (FAA)
Functional Needs Analysis (FNA)
Functional Solutions Analysis (FSA)

Dave Krueger
Chris Ryder
Contributions from Lee Kennedy and Bob Finlayson
JCIDS Process*

DOD Strategic Guidance

Joint Operations Concepts

Joint Operating Concepts

Joint Functional Concepts

Joint Integrating Concepts

Functional Area Analysis

Functional Needs Analysis

JCD

DOTLPF Analysis (Non-materiel Approaches)

Ideas for Materiel Approaches

Analysis of Materiel/Non-materiel Approaches

Alternative N

Alternative 2

Alternative 1

Post Independent Analysis

Integrated Architectures

CPD

CDD

ICD

DCR

*CJCSM 3170.01B
The Defense Acquisition Management Framework*

- Process entry at Milestones A, B, or C
- Entrance criteria met before entering phase
- Evolutionary Acquisition or Single Step to Full Capability

*DoDI 5000.2, 12 May 2003
Discussion Topics

- Perspective
- Introduction to the TBDA Case
- Functional area analysis (FAA)
- Functional needs analysis (FNA)
- Functional solutions analysis (FSA)
- Post independent analysis (PIA)
- Conclusion
JCIDS Process

DOD Strategic Guidance

Joint Operations Concepts

Joint Operating Concepts
Joint Functional Concepts
Joint Integrating Concepts

Functional Area Analysis

Functional Needs Analysis

JCD

DOTLPF Analysis (Non-materiel Approaches)

Ideas for Materiel Approaches

Analysis of Materiel/Non-materiel Approaches

Alternative N
Alternative 2
Alternative 1

Post Independent Analysis

Integrated Architectures

CPD

CDD

ICD

DCR

*CJCSM 3170.01B
Functional Area Analysis (FAA)

- Produces a list of capabilities across all functional areas necessary to achieve military objectives

- Capabilities
  - Operational tasks
  - Conditions
  - Standards (or measures of effectiveness)

- Input
  - National strategies, JOCs, JFCs, JICs, the Universal Joint Task List (UJTL)
  - Anticipated adversary capabilities
Defining the Problem Space
The Capability Context

- Understanding the Capability Enterprise
  - Environment
  - Enemy forces and systems
- Capability operations
- Capability operators (Warriors)
- Network requirements
- Capability command and control
  - Command authority
- Analysis of legacy Warfare Systems contributing to the Capability Object
- Preliminary Non-Materiel issues
  - DOTMLPF analysis primarily occurs during the FSA
Architecture Meta-Model

class Architecture Model

Architecture

Architecture Model

«DataType»

Physical

Model

Physical

Element

Dynamic

Element

Association

Architecture

Element

Architecture

View

«DataType»

Requirements

Model

«DataType»

Functional

Model

Traceability Matrix

Depicts
Capability Context  
(Repeat from Model Driven SE Section)
FAA Activity Diagram

- Collect Relevant Capability Artifacts
  - Environmental conditions, timeline, threat analysis
  - Information Gathering
  - Environmental conditions, timeline, threat analysis
  - The basis for Mission Level Use Cases and Scenarios.
  - These tasks are foundation of "To-Be" Enterprise Model.

- Perform Cross-Capability and Cross-Section Analysis
  - Understanding the legacy Capability "As Is" contribution to the needed "To-Be" capabilities.

- Analyze Operating Conditions
  - Includes preliminary Capability requirements (operational needs), actors (warriors), tasks (activities) and Capability Context.

- ID tasks for FNA
  - National strategies, JOC, JFC, JIC, UJTL, and the anticipated range of broad capabilities that adversaries might employ.

- FAA Activity Diagram Components
  - «DataType» JOIDS Document
  - «Capability» Capability As Is
    - Attribute 1..N
    - + Operation 1..N: ROV
  - «DataType» Joint Capability Area
  - «DataType» UCJT
    - Operational: OP-1..N
    - Strategic: National: SN-1..N
    - Strategic: Theater: ST-1..N
    - Tactical: TA-1..N
  - JOCIDS Document
  - «Capability» Capability As Is
    - Attribute 1..N
    - + Operation 1..N: ROV
  - Architecture Model

- «DataType» Capability Task List
  - Measure 1..N
  - Mission 1..N
  - Standard
  - Task 1..N
  - Traceability to National Strategy

- «DataType» UAJC
  - Operational: OP-1..N
  - Strategic: National: SN-1..N
  - Strategic: Theater: ST-1..N
  - Tactical: TA-1..N
**FAA Approach**

- **Joint Operations Concepts (JOpsC)**
  - Fully Integrated
  - Expeditionary
  - Networked
  - Decentralized
  - Adaptable
  - Decision superiority
  - Lethality

- **Joint Functional Concepts (JFC)**
  - Force application
  - Force protection
  - Focused logistics
  - Battlespace awareness
  - Command and control
  - Force management
  - Net centric
  - Joint training

- **Joint Integrating Concepts (JIC)**
  - Forceable Entry Ops
  - Undersea Superiority
  - Global Strike Ops
  - Sea-Basing Ops
  - Air & Missile Defense
  - JC2
  - Joint Logistics

- **Joint Operating Concepts (JOC)**
  - Homeland Security
  - Stability Operations
  - Strategic Deterrence
  - Major Combat Operations

- **Universal Joint Task List (UJTL)**
  - Strategic National (SN)
  - Strategic Theater (ST)
  - Operational (OP)
  - Tactical (TA)

- **Universal Navy Task List (UNTL)**
  - UJTL (Strategic & Operational)
  - Navy Tactical Task List (NTTL)

- **Applicable, Existing FAAs**
  - Battlespace Awareness FAA

- **Program Specific Documentation**

- **Other Input**
  - Operational personnel
  - Customers

- **Required Capabilities**

---

*NDIA*

*APL*
Joint Operations Concepts (JOpsC)

- An overarching description of how the future Joint Force will operate across the entire range of military operations

- Attributes
  - Fully Integrated
  - Expeditionary
  - Networked
  - Decentralized
  - Adaptable
  - Decision superiority
  - Lethality

Too general for specific FAA development
Joint Integrating Concepts (JICs)

- Description of how a Joint Force Commander will integrate capabilities to generate effects and achieve an objective
  - Forceable Entry Ops
  - Undersea Superiority
  - Global Strike Ops
  - Sea-Basing Ops
  - Air & Missile Defense
  - JC2
  - Joint Logistics

- Includes an illustrative CONOPS for a specific scenario and a set of distinguishing principles applicable to a range of scenarios
Joint Operating Concepts (JOCs)

- Operational-level description of how the Joint Force Commander will operate and a foundation for defining military capabilities

- Operational context for JFC and JIC development
  - Homeland Security (HLS)
  - Stability Operations (SO)
  - Strategic Deterrence (SD)
  - Major Combat Operations (MCO)
Joint Functional Concepts (JFCs)

- Describes how the joint force will perform military functions across the range of military operations
- Functional areas
  - Force application
  - Force protection
  - Focused logistics
  - Battlespace awareness
  - Command and control
  - Force management
  - Net centric
  - Joint training

Functional Capability Board (FCB) for each functional area
Functional Capability Boards (FCB)

- Responsible for organization, analysis, and prioritization of capability needs proposals within their functional areas

- Provide oversight and assessment throughout JCIDS process
  - Reduce redundant analyses
  - Ensure consistency in capability definitions
  - Ensure approaches consider a broad range of possibilities

- Provides context briefing to JROC
  - Where capability proposal fits within functional area

- Make recommendations on validation and approval

Identify appropriate FCB and involve them in the analyses!
Battlespace Awareness JFC Capabilities

- **Operational**
  - Command and control of BA assets
  - Execute collection
  - Exploitation and analysis
  - M&S, forecast
  - Manage knowledge

- **Enabling**
  - Integrate BA network
  - Infuse emergent technology
  - Recruit, retain, train

Several capabilities IDed for each capability category, e.g.,
- Surveillance
- Cross cue
- Employ human resources
- Employ open source resources
- Measure & monitor environmental conditions

Battlespace Awareness FAA (Draft)

- Defines Battlespace Awareness capabilities for each task and sub-task in each JOC
  - Homeland Security (HLS)
  - Stability Operations
  - Strategic Deterrence
  - Major Combat Operations (MCO)

1. Interdiction
   a. Kill 1st echelon forces
   b. Divert/delay follow-on forces
2. Ground operations
3. Air defense
4. Missile defense
5. JSEAD
6. Strike
7. Sea Strike Operations
8. Sea Shield Operations
9. Sea Basing Operations
10. Info Operations
11. Battlespace Awareness
12. Intent/I&W
13. I&W Specific Threat

Tasks and sub-tasks for other JOCs not shown
UJTL and UNTL

- **UJTL**: “The Universal Joint Task List (UJTL), when augmented with the Service task lists, is a comprehensive integrated menu of functional tasks, conditions, measures, and criteria supporting all levels of the Department of Defense in executing the National Military Strategy.”

- **UNTL**: “The UNTL tasks make up a comprehensive hierarchical structure. The UNTL task list is designed to be comprehensive while being mutually exclusive. When reviewing the levels of the hierarchy, the subordinate tasks will, in total, comprehensively, and without redundancy, define all activities involved in the next higher-level task.”
Universal Joint Task List (UJTL)*
Levels of War

- **(SN) Strategic level - National military tasks**
  - Accomplish objectives of national military strategy

- **(ST) Strategic level - Theater tasks**
  - Accomplish objectives of the theater and campaign strategy

- **(OP) Operational level tasks**
  - Accomplish objectives of subordinate campaigns and major operations

- **(TA) Tactical level tasks - include joint/interoperability tactical tasks and the applicable Service tasks**
  - Accomplish objectives of battles and engagements

* CJCSM 3500.04C, “Universal Joint Task List (UJTL),” 1 July 2002
### UNTL Example Hierarchy

To obtain, by various detection methods, information about the activities of an enemy or potential enemy or tactical area of operations. This task uses surveillance to systematically observe the area of operations by visual, aural, electronic, photographic, or other means. This includes development and execution of search plans.

<table>
<thead>
<tr>
<th>NTA</th>
<th>Navy Tactical</th>
</tr>
</thead>
<tbody>
<tr>
<td>NTA 1</td>
<td>Deploy Forces/Conduct Maneuver</td>
</tr>
<tr>
<td>NTA 2</td>
<td>Develop Intelligence</td>
</tr>
<tr>
<td>NTA 2.1</td>
<td>Plan and Direct Intelligence Operations</td>
</tr>
<tr>
<td>NTA 2.2</td>
<td>Collect Data and Intelligence</td>
</tr>
<tr>
<td>NTA 2.2.1</td>
<td>Collect Target Information</td>
</tr>
<tr>
<td>NTA 2.2.2</td>
<td>Collect Tactical Intelligence on Situation</td>
</tr>
<tr>
<td>NTA 2.2.3</td>
<td>Perform Tactical Reconnaissance and Surveillance</td>
</tr>
<tr>
<td>NTA 2.2.4</td>
<td>Assess Tactical Environment</td>
</tr>
<tr>
<td>NTA 2.3</td>
<td>Process and Exploit Collected Info/Intelligence</td>
</tr>
<tr>
<td>NTA 2.3.1</td>
<td>Conduct Technical Processing and Exploitation</td>
</tr>
<tr>
<td>NTA 2.3.2</td>
<td>Correlate Information</td>
</tr>
<tr>
<td>NTA 2.4</td>
<td>Produce Intelligence</td>
</tr>
<tr>
<td>NTA 2.5</td>
<td>Disseminate and Integrate Intelligence</td>
</tr>
<tr>
<td>NTA 3</td>
<td>Employ Firepower</td>
</tr>
<tr>
<td>NTA 4</td>
<td>Perform Logistics and Combat Service Support</td>
</tr>
<tr>
<td>NTA 5</td>
<td>Exercise Command and Control</td>
</tr>
<tr>
<td>NTA 6</td>
<td>Protect the Force</td>
</tr>
</tbody>
</table>

To associate and combine data on a single subject to improve the reliability or credibility of the information. This task includes collating information (identifying and grouping related items of information for critical comparison).
Identify Tasks for FNA

- Select Tasks from UJTL
- Select Tasks from FCB Portfolio
- Define Standards and Conditions for Each Task
  - Capability Task List
    - Task 1..N
  - Task Measures List
    - Measure 1..N

- UJTL:
  - Strategic: National : SN-1..N
  - Strategic: Theater : ST-1..N
  - Operational: OP-1..N
  - Tactical : TA-1..N
Identify Tasks

Joint Operations Concepts (JOpsC)
- Fully Integrated
- Expeditionary
- Networked
- Decentralized
- Adaptable
- Decision superiority
- Lethality

Joint Functional Concepts (JFC)
- Force application
- Force protection
- Focused logistics
- • Battlespace awareness
- Command and control
- Force management
- Net centric
- Joint training

Joint Integrating Concepts (JIC)
- Forceable Entry Ops
- Undersea Superiority
- Global Strike Ops
- Sea-Basing Ops
- Air & Missile Defense
- JC2
- Joint Logistics

Joint Operating Concepts (JOC)
- Homeland Security
- Stability Operations
- Strategic Deterrence
- Major Combat Operations

Universal Joint Task List (UJTL)
- Strategic National (SN)
- Strategic Theater (ST)
- Operational (OP)
- Tactical (TA)

Universal Navy Task List (UNTL)
- UJTL
- Navy Tactical Task List (NTTL)

Applicable, Existing FAAs
- Battlespace Awareness FAA

Program Specific Documentation

Other Input
- Operational personnel
- Customers

Required Capabilities

Identify tasks
## Select Tasks from UJTL & UNTL

<table>
<thead>
<tr>
<th>NTA</th>
<th>Navy Tactical</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>NTA 1</td>
<td>Deploy Forces/Conduct Maneuver</td>
<td>P</td>
</tr>
<tr>
<td>NTA 2</td>
<td>Develop Intelligence</td>
<td>Y</td>
</tr>
<tr>
<td>NTA 2.1</td>
<td>Plan and Direct Intelligence Operations</td>
<td>N</td>
</tr>
<tr>
<td>NTA 2.2</td>
<td>Collect Data and Intelligence</td>
<td>Y</td>
</tr>
<tr>
<td>NTA 2.2.1</td>
<td>Collect Target Information</td>
<td>Y</td>
</tr>
<tr>
<td>NTA 2.2.2</td>
<td>Collect Tactical Intelligence on Situation</td>
<td>Y</td>
</tr>
<tr>
<td>NTA 2.2.3</td>
<td>Perform Tactical Reconnaissance and Surveillance</td>
<td>Y</td>
</tr>
<tr>
<td>NTA 2.2.4</td>
<td>Assess Tactical Environment</td>
<td>Y</td>
</tr>
<tr>
<td>NTA 2.3</td>
<td>Process and Exploit Collected Info/Intelligence</td>
<td>Y</td>
</tr>
<tr>
<td>NTA 2.3.1</td>
<td>Conduct Technical Processing and Exploitation</td>
<td>Y</td>
</tr>
<tr>
<td>NTA 2.3.2</td>
<td>Correlate Information</td>
<td>Y</td>
</tr>
<tr>
<td>NTA 2.4</td>
<td>Produce Intelligence</td>
<td>Y</td>
</tr>
<tr>
<td>NTA 2.5</td>
<td>Disseminate and Integrate Intelligence</td>
<td>P</td>
</tr>
<tr>
<td>NTA 3</td>
<td>Employ Firepower</td>
<td>P</td>
</tr>
<tr>
<td>NTA 4</td>
<td>Perform Logistics and Combat Service Support</td>
<td>N</td>
</tr>
<tr>
<td>NTA 5</td>
<td>Exercise Command and Control</td>
<td>P</td>
</tr>
<tr>
<td>NTA 6</td>
<td>Protect the Force</td>
<td>P</td>
</tr>
</tbody>
</table>
**Select Tasks from FCB Portfolios**

<table>
<thead>
<tr>
<th>Battlespace Awareness Tasks</th>
<th>Applicable</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Command and Control of BA Assets</strong></td>
<td></td>
</tr>
<tr>
<td>Synchronize ISR with operations</td>
<td>Y</td>
</tr>
<tr>
<td>Task, dynamically re-task and monitor assets</td>
<td>N</td>
</tr>
<tr>
<td><strong>Execute Collection</strong></td>
<td></td>
</tr>
<tr>
<td>Surveillance</td>
<td>Y</td>
</tr>
<tr>
<td>Cross cue</td>
<td>Y</td>
</tr>
<tr>
<td>Employ human resources</td>
<td>Y</td>
</tr>
<tr>
<td>Employ open source resources</td>
<td>N</td>
</tr>
<tr>
<td>Measure and monitor environmental conditions</td>
<td>P</td>
</tr>
<tr>
<td><strong>Exploit and Analyze</strong></td>
<td></td>
</tr>
<tr>
<td>Recognize targets</td>
<td>Y</td>
</tr>
<tr>
<td>Distribute processing</td>
<td>N</td>
</tr>
<tr>
<td>Information fusion</td>
<td>Y</td>
</tr>
<tr>
<td>Enable analyst collaboration</td>
<td>N</td>
</tr>
<tr>
<td>Defeat denial and deception</td>
<td>P</td>
</tr>
<tr>
<td><strong>Model, Simulate, Forecast/Predict</strong></td>
<td></td>
</tr>
<tr>
<td>Predictive analysis</td>
<td>N</td>
</tr>
<tr>
<td>Integrate adversary and friendly information</td>
<td>N</td>
</tr>
<tr>
<td><strong>Manage Knowledge</strong></td>
<td></td>
</tr>
<tr>
<td>Smart pull/push information</td>
<td>Y</td>
</tr>
<tr>
<td>Share plan visibility</td>
<td>N</td>
</tr>
<tr>
<td>Content management</td>
<td>N</td>
</tr>
</tbody>
</table>
Define Conditions and Standards for Each Task

- **Conditions** - a variable of the environment that affects performance of a task
  - Physical: land, sea, air, space
  - Military: mission; forces; C3; intelligence; deployment, movement, and maneuver; firepower; protection; sustainment; threat; conflict
  - Civil: political policies, culture, economy,

- **Standard** - the minimum proficiency required in the performance of a task
  - Measure - Quantitative or qualitative basis for describing the quality of task performance
  - Criterion - A critical, threshold, or specified value of a measure

- **Sources**
  - UJTL/UNTL
  - Design Reference Mission (DRM)
  - Subject Matter Experts (SMEs)
Example Conditions From UNTL

- C 1.0 PHYSICAL ENVIRONMENT
  - C 1.1 LAND
    - C 1.1.1 Terrain
      - C 1.1.1.1 Terrain Relief
      - C 1.1.1.2 Terrain Elevation
      - C 1.1.1.3 Terrain Slope
      - C 1.1.1.4 Terrain Firmness
      - C 1.1.1.5 Terrain Traction
    - C 1.1.1.6 Vegetation
      Plants, trees, and shrubs. 
      *Descriptors*: Jungle (rainforest, canopied); Dense (forested); Light (meadow, plain); Sparse (alpine, semi-desert); Negligible (arctic, desert).
    - C 1.1.1.7 Terrain Relief features
      - C 1.1.2 Geological Features
      - C 1.1.3 Man-Made Terrain Features
    - Etc.
Example Measures

NTA 2.2.3 Perform Tactical Reconnaissance and Surveillance

To obtain, by various detection methods, information about the activities of an enemy or potential enemy or tactical area of operations. This task uses surveillance to systematically observe the area of operations by visual, aural, electronic, photographic, or other means. This includes development and execution of search plans.

<table>
<thead>
<tr>
<th>Units</th>
<th>Measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Days</td>
<td>From receipt of tasking, until reconnaissance/surveillance assets in place</td>
</tr>
<tr>
<td>Percent</td>
<td>Of collection requirements fulfilled by reconnaissance/surveillance assets</td>
</tr>
<tr>
<td>Percent</td>
<td>Of time able to respond to collection requirements</td>
</tr>
<tr>
<td>Hours</td>
<td>To respond to emergent tasking</td>
</tr>
<tr>
<td>Percent</td>
<td>Operational availability of tactical aircraft reconnaissance systems</td>
</tr>
<tr>
<td>Time</td>
<td>To exploit single tasked image collected after aircraft on deck</td>
</tr>
</tbody>
</table>
FAA Results

Description of the operational/tactical situation including the appropriate conditions

3 days from receipt of tasking, until reconnaissance/surveillance assets in place

FAA

<table>
<thead>
<tr>
<th>Mission Type 1</th>
<th>Standard 1.1.1</th>
<th>Standard 1.1.2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task 1.1</td>
<td>Standard 1.1.1</td>
<td>Standard 1.1.2</td>
</tr>
<tr>
<td>Task 1.2</td>
<td>Standard 1.2</td>
<td>Standard 1.3</td>
</tr>
<tr>
<td>Task 1.3</td>
<td>Standard 1.3</td>
<td></td>
</tr>
<tr>
<td>Task 1.4</td>
<td>Standard 1.4</td>
<td></td>
</tr>
<tr>
<td>Task 1.5</td>
<td>Standard 1.5.1</td>
<td>Standard 1.5.2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Mission Type 2</th>
<th>Standard 1.5.3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mission Type 3</td>
<td></td>
</tr>
<tr>
<td>Mission Type 4</td>
<td></td>
</tr>
</tbody>
</table>

Description of the task, e.g., “Perform tactical reconnaissance and surveillance”

6 hours to exploit single tasked image collected after aircraft on deck
FAA Artifacts

- First iteration of Architecture Model
  - Capability Requirements Model
  - Capability Context Diagram
    - Block Definition Diagram
  - Identification of actors within the Capability Enterprise
    - Block Definition Diagram
  - Capability tasks depicted in Use Case Diagram
    - And Activity Diagram as appropriate
FAA Artifacts

- First Iteration of Architecture Model
  - Initial information exchanges and data elements
    - Sequence Diagram
  - Tasks can be captured as SysML Blocks that include task standards
    - Task attributes (measures)
    - Results of Value (Post-Conditions that determine success/failure)

- Capability Task List
  - Capability tasks trace to model elements, including capability requirements
TBDA FAA

- Define the operational enterprise for the TBDA that will:
  - Establish measurable capability needs for targeting and surveillance
  - Specify the pertinent operations for targeting and surveillance that will capture the correct capability tasks
  - Comprehend the “Capability Enablers”, those “things” that provide pieces of the overall capability
    - Including initial review of legacy systems
    - As well as non-materiel contributors
  - Who are the beneficiaries of the capability
  - What are the potential data elements, information exchanges and interfaces
  - Verify that the capability is required and that it is captured correctly
TBDA Context

class TBDA Context

- TBDA Warrior Operates
- TBDA Logistics & Support Operatives Supports
- Planning Element Operated by
- Directs
- Command Authority Supports
- Directs
- Senses
- Sensed by

«Capability»

- TBDA

«System»

- Global Information Grid Supports
- Supported by
- Supported Strike System
- Supported by
- Supported by
- TBDA Target
- Sensed by
- Supports
TBDA Context

- At this point in the analysis, the TBDA:
  - Is under the direction of some Command Authority
  - Senses a generic class of targets
  - Interfaces with the Global Information Grid
  - Operated by Warriors
  - Supported and maintained by Warriors
- It is highly probable that the Capability Context will be modified throughout the FAA (and follow-on analysis phases)
TBDA Capability Elements
(The FAA Requirements Model)

- Four basic needs categories
  - Networking/Data Link
    - TBDA Information must “get somewhere”
  - Sensors
    - Some “thing” must capture TBDA information
  - Vehicle
    - The capability must be deployed within a defined battlespace
  - Supportability and Logistics
    - “Professionals think Logistics”

- Requirements can be captured using data bases and graphical tools
  - Requirements Traceability Matrix (RTM)
  - SysML Requirements Diagram
  - These artifacts must be “tightly coupled”
TBDA FAA Requirements Diagram

req TBDA Requirements

TBDA System

- Area of Coverage
  (from Vehicle)
- Range for Coverage
  (from Vehicle)
- Time of Flight
  (from Vehicle)
- Vehicle
  (from Vehicle)

- Field of View
  (from Sensors)
- Field of Regard
  (from Sensors)
- Number of Targets for Simultaneous Track
  (from Sensors)
- Range of Speeds for Target Tracking
  (from Sensors)
- Range of Speeds for Target Tracking
  (from Sensors)
- Sensor Range
  (from Sensors)
- Target Radar Cross Section
  (from Sensors)
- Target Resolution
  (from Sensors)

- Target Radar Cross Section
  (from Sensors)
- Target Resolution
  (from Sensors)

- Range of Coverage for Data Link
  (from Data Link)
- Data Transfer Rate
  (from Data Link)
- Mean Flight Hours Between Maintenance Actions
  (from Logistics and Supportability)
- Mean Time for Maintenance Action
  (from Logistics and Supportability)
- System Set-up Time
  (from Logistics and Supportability)
- Vehicle Launch and Landing
  (from Logistics and Supportability)

SysML diagrams supported by robust database
TBDA Fundamental Operations

SysML Use Case Diagram Captures Basic Functionality Performed by Actors
Fundamental Operations Function Tree

class TBDA Functional Hierarchy

- «Use Case» Conduct TBDA Mission
  - «Use Case» Develop the TBDA Surveillance Mission
  - «Use Case» Transport the TBDA
  - «Use Case» Collect, Process and Assess Surveillance Information
  - «Use Case» Disseminate the TBDA Information
  - «Use Case» Support and Sustain the TBDA
TBDA Functional Decomposition
(Develop the TBDA Mission)

- Collect Relevant Pre-flight Data
- Conduct Preliminary Mission Analysis
- Develop Communications Network
- Develop Sensor Plan
- Develop Vehicle Operations Plan
- Develop Integrated Pre-Mission Plan
- Download Mission Plan to TBDA Vehicle

- Operational Information
- TBDA Mission Tasking
TBDA Functional Analysis
(Initial Data Exchanges)

Mission Development

Command Authority

Mission Development Team

TBDA Supporting Elements

TBDA Warrior

Mission Tasking

Request Operational Information

Operational Information

Mission Plan Ready for Download to TBDA
SysML Diagrams Supporting Functional Analysis

- **Activity Diagram**
  - Depicts functional elements as activities
    - Activities create Data Elements that are consumed in subsequent activities or use cases
    - Team can assess initial data requirements
    - Data Element is most often the Result of Value (ROV)
  - Modified to show “who” is performing the activities via “swim lanes”

- **Sequence Diagram (aka Interaction Diagram)**
  - Depicts sequence of information exchanges
    - Sending and receiving nodes
  - Analysis team should be able to get an initial understanding of interface requirements

**Activities and Use Cases Trace to UNTL Tasks**
TBDA FAA Logical Elements

class TBDA Capability

«System»
Legacy System
- Legacy Attributes (1..N);
+ Legacy Operations (1..N)

«Capability»
TBDA

Applies
0..*

«Capability Enabler»
Non-Material

«Capability Enabler»
TBDA Support Element
«Capability Enabler»
TBDA Controls
«Capability Enabler»
TBDA Sensor
«Capability Enabler»
TBDA Vehicle
«Capability Enabler»
TBDA Communications
Even during the FAA, some “Capability Enablers” can be logically deduced
- As analysis progresses, the attributes for these Enablers will be defined as well as functionality assigned to those elements

Acknowledged that there are some non-materiel contributors

Initial review of legacy systems
- Further studied during FNA
TBDA FAA Architectural Model

- Captured the operational need through a SysML Requirements Diagram
  - Along with RTM
  - SE Method: Requirements Analysis

- Identified basic functionality
  - Will contribute to UJTL task assessment
  - Initially capture potential data elements and interfaces
  - Help generate Capability Task List for the FNA
  - SE Method: Functional Definition

- Identified Capability Enablers
  - Along with “first look” contributions by Legacy Systems and non-materiel elements
  - SE Method: Physical Definition

- Each of the basic functions and Capability Enablers must trace to the capability requirements listed in the Statement of Operational Need
  - SE Method: Design Validation
JCIDS Process

- DOD Strategic Guidance
- Joint Operations Concepts
- Joint Operating Concepts
- Joint Functional Concepts
- Joint Integrating Concepts
- Functional Area Analysis
- Functional Needs Analysis
- DOTLPF Analysis (Non-materiel Approaches)
- Ideas for Materiel Approaches
- Analysis of Materiel/Non-materiel Approaches
- Integrated Architectures
- Alternative N
- Alternative 2
- Alternative 1
- Post Independent Analysis

*CJCSM 3170.01B
Functional Needs Analysis (FNA)

- Assess current and programmed warfighting systems
  - Can they deliver the capabilities identified in the FAA
  - Uses conditions from FAA
  - Uses standards from the FAA as the “measuring stick”

- Output
  - List of capability gaps or shortfalls
    - Relative priority
    - Timeframe for required solutions
  - Identify redundancies in capabilities that reflect inefficiencies.

Determines gaps in planned capabilities
FNA Activities

- Activity/ Task Mapping
  - Functional decomposition including measurable results of value
  - Assignment of functions to logical elements
  - Refinement of capability measures
    - Forwarded to JROC for approval

- Resource allocation
  - Contribution of legacy Warfare System capabilities

- Trades Analyses
  - Cost, schedule and performance constraints
  - Task alternatives and weighting

- Interoperability assessment
  - Refinement of information exchanges and date elements
FNA Functional Analysis

- Expand functional analysis from FAA
- Identify data elements/data attributes
- Consider interface elements such as communications links
- Quantify measurable use case results of value
- Functional contributions of legacy systems and subsystems
Functional Analysis with Use Cases

- **Use Case**
  - Sequence of events that returns a measurable Result of Value (Booch, Rumbaugh, Jacobson)
  - Captures
    - Actors (Warriors performing the activities)
      - Roles, not specific individuals or commands
    - Activities (operations)
    - Data objects
      - Created and consumed by the activities
      - Information elements that are exchanged between the Operational Nodes
    - Any other relevant references such as UJTL tasks

Functional Analysis is basic to the Systems Engineering Method
FNA Activity Diagram

- **FNA Activity Diagram**: Diagram illustrating the process of FNA (Functional Needs Analysis).
  - **Act**: Perform FNA
  - **Capability Task List**:
    - Mission: 1..N
    - Task: 1..N
    - Standard
    - Measure: 1..N
    - Traceability to National Strategy
  - **Capability Gap List**:
    - Capability Gap: 1..N
  - **Capability Task Weight**
  - **Resource List**:
    - Financial Resources: Money
    - Warfare Systems
    - Support Systems
  - **Functional Model**
  - **Use Case**: 1.*
  - **Capability Measure**
  - **Use Case ROV**
  - **Capability Element**
    - Weight
    - Risk
  - **Architecture Model**
  - **Capability Attribute**
  - **JROC Approved**
  - **Architecture Model**

- **Assess Capability Gaps, Overlaps and operational problem(s)**

- **Analyze potential new Functional Areas for problem or solution**

- **Formulate key attributes for capability development of Measures of Effectiveness**

- **Identify Key Architectural Elements affected by new capability**

- **From FAA: Architecture Model**
### Example FNA Methodology

1. Assign weights based on relative importance

2. Select alternatives (screen based on FAA)

3. Evaluate performance of each alternative for each mission, task, and standard

4. Scores less than one indicate capability gaps

<table>
<thead>
<tr>
<th>FAA</th>
<th>Weight</th>
<th>Alternative A</th>
<th>Alternative B</th>
<th>Alternative C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mission Type 1</td>
<td>0.30</td>
<td>0.79</td>
<td>0.79</td>
<td>0.69</td>
</tr>
<tr>
<td>Task 1.1</td>
<td>0.10</td>
<td>0.90</td>
<td>0.80</td>
<td>0.85</td>
</tr>
<tr>
<td>Standard 1.1.1</td>
<td></td>
<td>0.85</td>
<td>0.90</td>
<td>0.60</td>
</tr>
<tr>
<td>Standard 1.1.2</td>
<td>0.05</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Task 1.2</td>
<td>0.15</td>
<td>0.90</td>
<td>0.70</td>
<td>0.70</td>
</tr>
<tr>
<td>Standard 1.2</td>
<td></td>
<td>0.80</td>
<td>0.80</td>
<td>0.80</td>
</tr>
<tr>
<td>Task 1.3</td>
<td>0.30</td>
<td>0.50</td>
<td>0.50</td>
<td>0.40</td>
</tr>
<tr>
<td>Standard 1.3</td>
<td>0.10</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Task 1.4</td>
<td>0.05</td>
<td>0.70</td>
<td>0.70</td>
<td>0.50</td>
</tr>
<tr>
<td>Standard 1.5.1</td>
<td></td>
<td>0.70</td>
<td>0.70</td>
<td>0.50</td>
</tr>
<tr>
<td>Task 1.5</td>
<td>0.20</td>
<td>0.75</td>
<td>0.95</td>
<td>0.70</td>
</tr>
<tr>
<td>Standard 1.5.2</td>
<td>0.05</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Standard 1.5.3</td>
<td>0.05</td>
<td>0.95</td>
<td>0.90</td>
<td>0.50</td>
</tr>
<tr>
<td>Mission Type 2</td>
<td>0.30</td>
<td>0.90</td>
<td>0.70</td>
<td>0.80</td>
</tr>
<tr>
<td>Mission Type 3</td>
<td>0.30</td>
<td>0.80</td>
<td>0.90</td>
<td>0.70</td>
</tr>
<tr>
<td>Mission Type 4</td>
<td>0.10</td>
<td>0.70</td>
<td>0.90</td>
<td>0.60</td>
</tr>
<tr>
<td>Composite Score</td>
<td></td>
<td>0.82</td>
<td>0.81</td>
<td>0.72</td>
</tr>
</tbody>
</table>
FNA Artifacts

- Next iteration of Architecture Model including
  - Definition of Capability Elements
    - Model as Blocks including attributes, parameters and constraints
  - Definition of legacy systems that contribute to the capability
  - Capturing functional tasks as use cases with ROVs
  - Assignment of functional tasks to Capability (logical) Elements
  - Use cases assigned to Capability Elements as Block operations
- Capability measures forwarded to JROC for approval
- Requirements Traceability Matrix
To what extent does the Legacy attributes and operations satisfy the capability need, including UJTL tasks?
FNA for Capability Enablers

Existing subsystems/components with suitable performance measures may provide some capability requirements, but will need to be integrated into an overall solution.
Key Architectural Elements

- In the FNA, architectural elements are still abstractions (i.e. capability enablers) of real systems
- Architectures include behaviors, relationships AND rules for “rules governing their design over time” (DoDAF)
  - FNA is the time to consider the “possible” with regard to applications for the capability enablers
    - Including the standards that govern the application
- Example, TBDA Communications
  - Tactical application places limits on size, weight, range of operations
  - Logical conclusion (after analysis): Comm Applications limited to Link-16, UHF/ VMF and CDL
    - Standardized interfaces exist for those applications
    - Modeled using SysML Internal Block Diagram
TBDA FNA Communications Architecture

composite structure Comm_Interfaces

<table>
<thead>
<tr>
<th>«System»</th>
<th>«Capability Enabler»</th>
</tr>
</thead>
<tbody>
<tr>
<td>Global Information Grid</td>
<td>TBDA Communications</td>
</tr>
</tbody>
</table>

- Link-16/ Mil-Std-6016
- UHF/ VMF/ Mil-Std-6017
- Common Data Link

Port: Interface defined by Standard
TBDA FNA Model

- Define operational functionality and assign that functionality to the logical elements, i.e. Capability Enablers
- Define capability attributes with suitable measures of effectiveness
  - Assign measures to key architectural elements
- Do these elements satisfy the identified capability gaps?
  - If not, refine
JCIDS Process

- DOD Strategic Guidance
- Joint Operations Concepts
- Joint Operating Concepts
- Joint Functional Concepts
- Joint Integrating Concepts
- Functional Area Analysis
- Functional Needs Analysis
- DOTLPF Analysis (Non-materiel Approaches)
- Ideas for Materiel Approaches
- Analysis of Materiel/Non-materiel Approaches
- Alternative N
- Alternative 2
- Alternative 1
- Integrated Architectures
- Post Independent Analysis

*CJCSM 3170.01B
Functional Solution Analysis (FSA)

- Operational assessment of all approaches to solving the capability gaps identified in the FNA
  - Non-materiel solutions
  - Materiel solutions (in priority order)
    - Product improvements to existing materiel or facilities
    - Adoption of interagency or foreign materiel solutions
    - Initiation of new materiel programs

- Basis for ICD

Transition from Problem Space to Solution Space
FSA Activities

- Define the Solution Space
  - Trace possible solutions to satisfactory “solve the problem”
- Conduct the DOTMLPF analysis
  - Model DOTMLPF Elements as Blocks that include attributes and operations
- Refine Use Cases and appropriate ROVs after DOTMLPF factored into Solution Space
- Analyze potential material solutions, i.e. Warfare Systems
  - Model Warfare Systems as logical elements and assign use cases – Assignment of functionality to physical elements
FSA Activities

- Analysis of Material Alternatives (AMA)
  - Analyze Capability Gap and range of military operations
  - Assess operational risk and DOTMLPF implications
  - Assess material impact to functional areas
- Program Independent Analysis (PIA)
  - Ensure the list of approaches with the potential to deliver the capability identified in the FAA and FNA is complete
FSA Activity Diagram

The AMA is significant enough to have a separate Use Case to denote the series of activities.

Updated list of Material and non-material approaches from AMA.

Updated Model factoring in material and non-material elements.

Includes Capability attributes, measures and constraints.

Updated list of Material and non-material approaches from AMA.

Updated Model factoring in material and non-material elements.

Includes Capability attributes, measures and constraints.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.

Updated Model factoring in material and non-material elements.
FSA: Non-Materiel Solutions

- Can FNA capability gaps be mitigated via a non-materiel solution (DOT_LPFL)
  - Doctrine
  - Organization
  - Training
  - Leadership/education
  - Personnel
  - Facilities

Generally a qualitative assessment
DOTMLPF as a Logical Element

class DOTMLPF

- «Capability» Capability Object

«DataType» DOTMLPF Element

- «DataType» Doctrine
- «DataType» Organization
- «DataType» Training
- «DataType» Leadership
- «DataType» Personnel
- «DataType» Facility

Contributes to

slide 126
Doctrine “Block”

class Doctrine

«DataType»
Doctrine

Attribute

Force
Data
Policy
Material

Operation

Employment
Sustainment
Comms
Tactics

Intel
FSA: Materiel Solutions
Analysis of Materiel Approaches (AMA)

- Assess potential materiel solutions to FNA capability gaps
  - Performance
  - Cost
  - Risk

- Some similarity to the Analysis of Alternatives (AoA)
  - Less rigorous
  - Less specific
**Notional FSA Results**

<table>
<thead>
<tr>
<th>Approach</th>
<th>Effectiveness</th>
<th>Cost</th>
<th>Risk</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>◦</td>
<td>$</td>
<td>◦</td>
</tr>
<tr>
<td>B</td>
<td>◦</td>
<td>$$</td>
<td>◦</td>
</tr>
<tr>
<td>C</td>
<td>●</td>
<td>$</td>
<td>◦</td>
</tr>
<tr>
<td>D</td>
<td>●●●●●</td>
<td>$$$$</td>
<td>●</td>
</tr>
</tbody>
</table>

Potential non-materiel and material approaches
FSA Artifacts

- Final Architecture Model
  - “To-Be” model
  - Material elements that perform functional operations that provide capability
    - Measurable Results of Value
- JCIDS Document
  - ICD for Pre-Milestone A
  - CDD and CPD for every capability upgrade
- Concept of Operations (CONOPS)
Concept of Operations Document

- The “CONOPS” is not a required JCIDS artifact, however
  - CONOPS document is a critical interface between the operational and the engineering communities
  - Provides potential developers the framework on how the capability will be applied in the operational environment
- Services may require some form of CONOPS
  - USAF Enabling Concept
FSA TBDA “To-Be” System

- **MOE**
  - Non-Material Contributors 1..N:
    - Develop the TBDA Surveillance Mission()
    - Transport the TBDA()
    - Collect, Process and Assess Surveillance Information()
    - Disseminate the TBDA Information()
    - Support and Sustain the TBDA()

  **MOE**
  - Time of Flight: > Three Hours
  - Range for Coverage: > 300 KM
  - Range of Coverage for Comm: > 400 KM
  - Area of Coverage: > 62,500 KM
    + Execute Vehicle Transport Functions()

  **MOE**
  - Data Link Effective Range: 400 KM
  - Bandwidth/Wave Form: Link-16/ UHF-VMF
  - Data Transfer Rate:
    + Disseminate TBDA Information()
    + Interface with GIG()

  **MOE**
  - Launch and Landing Range: int <= 100 Meters
  - Set Up Time: <= 30 Minutes
  - Mean Maintenance Time: <= 2 hours when ...
  - MTBMA: => 48 Flight Hours
    + Support and Sustain the TBDA () : void

**Subsystem**
- TBDA Air Vehicle
  - Execute Vehicle Transport Functions()

**Subsystem**
- TBDA Ground Control Station
  - Develop the TBDA Surveillance Mission()
  - Control TBDA Air Vehicle()
  - Process TBDA Information()

**Subsystem**
- TBDA Sensor System
  - Field of View (Staring): => 30 Degrees
  - Sensor Field of Regard: => 60 Degrees
  - Sensor Range Minimum: <= 5 KM
  - Sensor Range Maximum: <= 30 KM
  - Minimum RCS: = 2 Square Meters
  - Range of Target Speeds: = 5 to 75 MPH
    + Capture Surveillance Information(

**DataType**
- Capability Gap List
  - Capability Gap 1..N

**Trace**
- from Vehicle
- from Communications
- from Logistics and Supportability
TBDA Development Options

Emphasis is still the Capability with options to pursue during concept evaluation and risk reduction phase (Post MS A)
JCIDS Process

DOD Strategic Guidance

Joint Operations Concepts

Joint Operating Concepts
Joint Functional Concepts
Joint Integrating Concepts

Functional Area Analysis

Functional Needs Analysis

DOTLPF Analysis (Non-materiel Approaches)

Ideas for Materiel Approaches

Analysis of Materiel/Non-materiel Approaches

Alternative N
Alternative 2
Alternative 1

Integrated Architectures

Post Independent Analysis

*CJCSM 3170.01B
Post Independent Analysis

- Final independent review of FAA, FNA, and FSA
  - Not the same people who conducted the analyses
  - Ensure...
    - Analyses were thorough
    - Potential solutions are reasonable
    - Potential solution set is complete
FAA, FNA, FSA, PIA Output

- There is no capability gap

- Capability gap can be addressed by change to:
  - DOT_LP - Doctrine, organization, training, leadership/education, personnel, and facilities
  - DCR - DOTLPF Change Request

- A materiel solution is required
  - Initial Capabilities Document (ICD)
Conclusion on Functional Analyses

- FAA, FNA, FSA, and PIA are import steps to identify, assess, and prioritize joint military capability needs
  - FAA – required capabilities
  - FNA – gaps in planned capabilities
  - FSA – potential solutions

- Conducted through a combination of quantitative and qualitative analyses

- Involve all stakeholders in the process
Tutorial Wrap-up

- JCIDS is an engineering intensive process!
- The Systems Engineering Method is appropriate for guiding the JCIDS analyses in every phase of the capability/system life cycle
  - Ensures traceability system functionality back to requirements
- Model Driven SE enables the JCIDS Team to fully understand what they are doing
  - SE Models provide the basis for the system’s architecture and all architectural views
  - SE Model is a living entity that transitions from JCIDS Team to Development Team
    - Today’s “To-Be” model becomes tomorrow’s “As-Is”
Tutorial Wrap-up

- JCIDS Functional Analyses, including AMA and PIA, are essential SE functions
  - Each phase, from FAA through FSA, better quantifies the degree of “materialization”
    - Including non-materiel capability contributors
- OMG SysML is most appropriate for modeling capabilities from early conceptualization to system design
  - Either OO or Traditional Structured methods

Good SE ← Effective JCIDS
Integrating Systems Engineering with Earned Value Management

NDIA Systems Engineering Conference
San Diego, CA
October 23, 2006

Paul J. Solomon, PMP
Performance-Based Earned Value®
Paul.Solomon@PB-EV.com

© 2006 Paul J. Solomon
Agenda

• Federal Policy and Guidance, Customer Expectations, EVM Limitations
• Newest Standards, Models, and Best Practices
• Project Management with Performance-Based Earned Value® (PBEV℠)
• Implementing PBEV into Your Project
• IT/Software Progress Measurement Issues
• Implementing Better Acquisition Management into Your Project
Copyright© 2006 by Paul Solomon.
This material contains excerpts from the book, *Performance-Based Earned Value*.®
The excerpts were reprinted courtesy of John Wiley & Sons, Inc. and the IEEE Computer Society Press. *Performance-Based Earned Value*® was written by Paul Solomon and Ralph Young and copyrighted by the IEEE, 2007.
Project Management
Shortfalls

- Inadequate early warning
- Schedules, EV overstate true progress
- Remaining work underestimated
Does EVMS Really Integrate?

EVMS

WBS

COST

SCHEDULE

TECHNICAL PERFORMANCE

RISK

© 2006 Paul J. Solomon
EVM data will be reliable and accurate only if:

• The right base measures of technical performance are selected and
• Progress is objectively assessed.
Federal Policy and Guidance, Customer Expectations, EVM Limitations
## Government Pays But Fails to Get Desired Outcomes

<table>
<thead>
<tr>
<th>GAO Report</th>
<th>Title</th>
<th>Findings and Recommendations</th>
</tr>
</thead>
<tbody>
<tr>
<td>06-66</td>
<td>Defense Acquisitions: DOD Paid Billions in Award and Incentive Fees</td>
<td>• Contractors not held accountable for achieving desired outcomes:</td>
</tr>
<tr>
<td></td>
<td>Regardless of Acquisition Outcomes</td>
<td>o Cost goals</td>
</tr>
<tr>
<td></td>
<td></td>
<td>o Schedule goals</td>
</tr>
<tr>
<td></td>
<td></td>
<td>o Desired capabilities</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Programs do not capture early on the requisite knowledge needed to effectively manage program risks</td>
</tr>
<tr>
<td>06-391</td>
<td>Defense Acquisitions: Assessments of Major Programs</td>
<td>DOD needs to change its requirements and budgeting processes to get desired outcomes from the acquisition process</td>
</tr>
</tbody>
</table>

(a) Government Accountability Office  
© 2006 Paul J. Solomon
<table>
<thead>
<tr>
<th>GAO Report</th>
<th>Title</th>
<th>Findings and Recommendations</th>
</tr>
</thead>
<tbody>
<tr>
<td>06-250</td>
<td>Information Technology: Improve the Accuracy and Reliability of Investment Information</td>
<td>2. If EVM not implemented effectively, decisions based on inaccurate and potentially misleading information 3. Agencies not measuring actual vs. expected performance in meeting IT performance goals.</td>
</tr>
<tr>
<td>GAO Report</td>
<td>Title</td>
<td>Findings and Recommendations</td>
</tr>
<tr>
<td>------------</td>
<td>----------------------------------------------------------------------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>04-722</td>
<td>Information Technology: DOD’s Acquisition Policies and Guidance</td>
<td>Best Practices and Controls:</td>
</tr>
<tr>
<td></td>
<td>DOD Systems Modernization</td>
<td>• Ensure that <em>requirements</em> are</td>
</tr>
<tr>
<td>06-215</td>
<td></td>
<td>traceable, verifiable, and controlled.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Trace requirements to system design specifications and testing documents.</td>
</tr>
</tbody>
</table>
|            |                                                                      | • Continually measure an acquisition’s *performance*, cost, and schedule against approved baselines.
## GAO Best Practices

<table>
<thead>
<tr>
<th>GAO Report No.</th>
<th>Title</th>
<th>Findings and Recommendations</th>
</tr>
</thead>
</table>
| 06-110        | Best Practices: Better Support of Weapons System „Needed to Improve Outcomes | Best Practice Controls:  
• Complete subsystem and system design reviews  
• Demonstrate with prototype that design meets requirements  
• Agreement that drawings are complete and producible |
| 06-368        | Defense Acquisitions: Major Weapon Systems Continue to Experience Cost and Schedule Problems |                                                                                             |
U.S. Federal Policy on SE
DOD Policy & Guidance on SE

Policy for Systems Engineering in DOD Policy 2/20/04

Defense Acquisition Guidebook (DAG) 10/8/04

Systems Engineering Plan Preparation Guide (SEP) 2/10/06

WBS Handbook, Mil-HDBK-881A (WBS) 7/30/05

Integrated Master Plan (IMP) & Integrated Master Schedule Preparation & Use Guide (IMS) 10/21/05

Risk Management Guide for DOD Acquisition (RISK) Aug. 06
DOD Policy on Award Fees (1)

- Link award fees to desired program outcomes
- Tie award fees to
  - Identifiable interim outcomes
  - Discrete events or milestones
    - Timely completion of:
      - Preliminary design review (PDR)
      - Critical design review (CDR)
    - Assessment of interim progress towards PDR, CDR
- Provisions explain how a contractor’s progress will be evaluated

1: OUSD (AT&L) Memo: Award Fee Contracts, 3/29/06

© 2006 Paul J. Solomon
<table>
<thead>
<tr>
<th>Policy or Guide (1 of 3)</th>
<th>Policy</th>
<th>DAG</th>
<th>SEP</th>
<th>WBS</th>
<th>IMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Develop SEP</td>
<td>P</td>
<td>4.2.3.2</td>
<td>1.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Technical reviews:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Event-driven timing</td>
<td>P</td>
<td>4.5.1</td>
<td>3.4.4</td>
<td>3.2.3.1</td>
<td>2.3, 3.3.2</td>
</tr>
<tr>
<td>Success criteria</td>
<td>P</td>
<td>4.5.1</td>
<td>3.4.4</td>
<td>3.2.3.1</td>
<td></td>
</tr>
<tr>
<td>Assess technical maturity</td>
<td></td>
<td>4.5.1</td>
<td>3.4.4</td>
<td>3.2.3.1</td>
<td></td>
</tr>
<tr>
<td>Integrate SEP with:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IMP</td>
<td>4.5.1</td>
<td>3.4.5</td>
<td></td>
<td>1.2, 2.3</td>
<td></td>
</tr>
<tr>
<td>IMS</td>
<td>4.5.1</td>
<td>3.4.5</td>
<td></td>
<td>1.2, 2.3</td>
<td></td>
</tr>
<tr>
<td>Technical Performance Measures (TPM)</td>
<td>4.5.1</td>
<td>3.4.4</td>
<td>1.2, 2.3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EVM</td>
<td>4.5.1</td>
<td>3.4.5</td>
<td>1.2, 2.3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### DOD Guides

**Integrate WBS with requirements specification, statements of work (SOW), IMP, IMS, and EVMS**

- 2.2.3, 3.2.3.3
- 3.4.3

**TPMs to compare actual vs. plan:**
- Technical development
- Design maturity

- 4.5.5
- 3.4.4
- 3.3.2

**TPMs to report degree to which system requirements are met:**
- Performance
- Cost
- Schedule

- 4.5.5
- 3.4.4

**Standards and models to apply SE**

- 4.2.2
- 4.2.2.1

**Institute requirements management and traceability**

- 4.2.3.4
- 3.4.4
## Contractor:
- Incorporate **risk mitigation** activities into the **IMS** and budgets
- Use **IMS** and **EVM** to monitor progress against **risk plans**

<table>
<thead>
<tr>
<th>Guides (3 of 3)</th>
<th>IMS</th>
<th>Risk</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Include quantified risk assessments in Estimate at Completion (EAC)</td>
<td>3.5</td>
<td>8.6.6</td>
</tr>
</tbody>
</table>
OMB

- Office of Management and Budget (OMB)
- Circular No. A-11, Section 300
  - Planning, Budgeting, Acquisition and Management of Capital Assets
- Section 300-5
  - *Performance-based* acquisition management
  - Based on EVMS standard
  - Measure progress towards milestones
    - Cost
    - *Capability to meet specified requirements*
    - Timeliness
    - *Quality*
Newest Standards, Models, and Best Practices
Quality

Quality = technical performance:

*Ability* (current or projected)

of a set of inherent characteristics of a product

Product component or

Process

to *fulfill requirements* of *customers*

CMMI definition
SE Life Cycle Work Products
IEEE 1220

Requirements Analysis

- Requirements Baseline
- Requirements Validation
  - Validated Requirements Baseline
    - Functional Analysis
      - Functional Architecture
        - Functional Verification
          - Verified Functional Architecture
            - Synthesis
              - Physical Architecture
                - Design Verification
                  - Verified Physical Architecture

  - Functional trade studies and assessments
    - Functional trade studies and assessments
      - Design trade studies and assessments

© 2006 Paul J. Solomon
<table>
<thead>
<tr>
<th>DAG Technical Review</th>
<th>DAG Baseline</th>
<th>DAG</th>
<th>IEEE 1220</th>
</tr>
</thead>
<tbody>
<tr>
<td>System Functional Review</td>
<td>System Functional Baseline</td>
<td>4.3.3.4.3</td>
<td>Validated Requirements Baseline</td>
</tr>
<tr>
<td>Preliminary Design Review</td>
<td>System Allocated Baseline</td>
<td>4.3.3.4.4</td>
<td>Verified Physical Architecture</td>
</tr>
<tr>
<td>Critical Design Review</td>
<td>System Product Baseline</td>
<td>4.3.3.4.5</td>
<td>Verified Physical Architecture</td>
</tr>
<tr>
<td>Production Readiness Review</td>
<td>System Product Baseline</td>
<td>4.3.3.9.3</td>
<td>Verified Physical Architecture</td>
</tr>
</tbody>
</table>
Measure the *allocated requirements* to determine:

- Development maturity vs. plan
- Indicated Quality
## Requirements Progress

<table>
<thead>
<tr>
<th>IEEE 1220</th>
<th>EIA-632</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.8.1.5 Performance-based progress measurement</td>
<td>4.2.1 Planning process, Req. 10: Progress against requirements</td>
</tr>
<tr>
<td>6.8.6 Track product ... metrics</td>
<td></td>
</tr>
<tr>
<td>6.8.1.5 d) Assess</td>
<td>Assess progress ...</td>
</tr>
<tr>
<td>• Development maturity to date</td>
<td>• Compare system definition</td>
</tr>
<tr>
<td>• Product’s ability to satisfy requirements</td>
<td>Against requirements</td>
</tr>
<tr>
<td>6.8.6 Product metrics...at pre-established control points enable:</td>
<td>a) Identify product metrics and expected values</td>
</tr>
<tr>
<td>• Overall system quality evaluation</td>
<td>• Quality of product</td>
</tr>
<tr>
<td>• Comparison to planned goals and targets</td>
<td>• Progress towards satisfying requirements</td>
</tr>
</tbody>
</table>

© 2006 Paul J. Solomon
## Technical Performance Measures (TPM)

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TPMs</strong> are key to progressively assess technical progress</td>
<td><em>Predict</em> future value of <em>key technical parameters</em> of the end system based on current assessments</td>
</tr>
<tr>
<td>• Establish <em>dates</em> for</td>
<td><em>Planned value</em> profile is time-phased achievement projected</td>
</tr>
<tr>
<td>– Checking Progress</td>
<td>• <em>Achievement to date</em></td>
</tr>
<tr>
<td>– Meeting full conformance to requirements</td>
<td>• <em>Technical milestone where TPM evaluation is reported</em></td>
</tr>
</tbody>
</table>
• How well a system is achieving performance requirements
• Use actual or predicted values from:
  – Engineering measurements
  – Tests
  – Experiments
  – Prototypes
• Examples:
  – Payload
  – Response time
  – Range
  – Power
  – Weight
Use TPMs as a base measure of EV

TPM Planned Value Profile

Tolerance Bands

Percent Required Value

Achievement to Date
## INCOSE Warning on TPM

### TPM per INCOSE Systems Engineering Handbook

<table>
<thead>
<tr>
<th>• TPMs express the objective performance requirements.</th>
</tr>
</thead>
<tbody>
<tr>
<td>• <strong>Without</strong> TPM</td>
</tr>
<tr>
<td>o Project manager could fall into the trap of relying on cost and schedule status alone</td>
</tr>
<tr>
<td>o Can lead to a product developed on schedule and within cost that does not meet all key requirements.</td>
</tr>
<tr>
<td>• Periodic recording of status of each TPM</td>
</tr>
<tr>
<td>o Provides continuing verification of degree of anticipated and actual achievement of technical parameters.</td>
</tr>
</tbody>
</table>

© 2006 Paul J. Solomon
Success Criteria and Work Products Per SE Standards
### IEEE 1220, (6.2): Success Criteria

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>• Represents identified customer expectations</td>
<td></td>
</tr>
<tr>
<td>• Represents constraints</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>– Project</td>
</tr>
<tr>
<td></td>
<td>– Enterprise</td>
</tr>
<tr>
<td></td>
<td>– External</td>
</tr>
<tr>
<td>• Stays within constraints.</td>
<td></td>
</tr>
</tbody>
</table>
## Validated Requirements Baseline

<table>
<thead>
<tr>
<th>IEEE 1220, (6.1, 6.2): Work Products</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Customer expectations</td>
</tr>
<tr>
<td>• Project, enterprise and external constraints</td>
</tr>
<tr>
<td>• Operational scenarios</td>
</tr>
<tr>
<td>• Measures of effectiveness (MOE)</td>
</tr>
<tr>
<td>• Interfaces</td>
</tr>
<tr>
<td>• Functional requirements</td>
</tr>
<tr>
<td>• Measures of performance (MOP)</td>
</tr>
<tr>
<td>• Modes of operation</td>
</tr>
<tr>
<td>• Design characteristics</td>
</tr>
<tr>
<td>• Human factors</td>
</tr>
<tr>
<td>• Documented trade-offs</td>
</tr>
</tbody>
</table>
**Verified Functional Architecture**

<table>
<thead>
<tr>
<th>IEEE 1220, (6.3, 6.4): Work Products</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Functional context analysis</td>
</tr>
<tr>
<td>– Functional behaviors</td>
</tr>
<tr>
<td>– Functional interfaces</td>
</tr>
<tr>
<td>– Allocated performance requirements</td>
</tr>
<tr>
<td>• Functional decomposition</td>
</tr>
<tr>
<td>– Subfunctions</td>
</tr>
<tr>
<td>– Subfunction states and modes</td>
</tr>
<tr>
<td>– Data and control flows</td>
</tr>
<tr>
<td>– Functional failure modes and effects</td>
</tr>
</tbody>
</table>
**Verified Functional Architecture**

<table>
<thead>
<tr>
<th><strong>IEEE 1220, (6.4): Success Criteria</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>• Meets requirements of <em>validated requirements baseline</em></td>
</tr>
<tr>
<td>• System functions decomposed to <em>lower-level functions</em> that shall be satisfied by elements of the system design</td>
</tr>
<tr>
<td>– Subsystems</td>
</tr>
<tr>
<td>– Components</td>
</tr>
<tr>
<td>– Parts</td>
</tr>
<tr>
<td>• Requirements upwardly traceable to the validated requirements baseline</td>
</tr>
</tbody>
</table>
Success Criteria of Technical Reviews

<table>
<thead>
<tr>
<th>IEEE 1220, Preliminary design stage</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.2.4.1 Subsystem reviews</td>
</tr>
<tr>
<td>a. Subsystem definition</td>
</tr>
<tr>
<td>• Mature</td>
</tr>
<tr>
<td>– Meet SE milestone criteria</td>
</tr>
<tr>
<td>a. Component allocations and specifications</td>
</tr>
<tr>
<td>– Provide a sound subsystem concept</td>
</tr>
<tr>
<td>c. Subsystem risks assessed and mitigated</td>
</tr>
<tr>
<td>d. Trade-study data...substantiate that subsystem requirements are achievable</td>
</tr>
</tbody>
</table>
IEEE 1220, 5.2 Preliminary design stage

5.2.4.2 System review

- After completion of subsystem reviews
- Does total system approach to detailed design satisfy the system baseline?
- Unacceptable risks are mitigated
- Issues for all subsystems, products, and life cycle processes are resolved
- Accomplishments and plans warrant continued development effort.
### Success Criteria of Technical Reviews

<table>
<thead>
<tr>
<th>IEEE 1220, Detailed design stage (Critical Design Review (CDR))</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.3.4.1 Component reviews</td>
</tr>
<tr>
<td>a. Detailed component definition...<strong>mature</strong>...meet</td>
</tr>
<tr>
<td>• MOE</td>
</tr>
<tr>
<td>• MOP criteria;</td>
</tr>
<tr>
<td>c. <strong>Risks</strong>...<strong>mitigated to</strong>...support</td>
</tr>
<tr>
<td>fabrication, assembly, integration, test.</td>
</tr>
<tr>
<td>d. <strong>Trade-study data</strong> ...<strong>substantiate</strong> that detailed</td>
</tr>
<tr>
<td>component <strong>requirements</strong> are <strong>achievable</strong></td>
</tr>
</tbody>
</table>
Success Criteria of Technical Reviews

<table>
<thead>
<tr>
<th>IEEE. 1220, Detailed design stage</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>5.3.4.3 System review</strong></td>
</tr>
<tr>
<td>• After component and subsystem reviews</td>
</tr>
<tr>
<td>• Does detailed design <em>satisfy the system baseline?</em></td>
</tr>
<tr>
<td>• Unacceptable <em>risks</em> are <em>mitigated</em></td>
</tr>
<tr>
<td>• <em>Issues</em> for all subsystems, products, and life cycle processes are <em>resolved</em></td>
</tr>
</tbody>
</table>
IEEE 1220, (6.6): Success Criteria

<table>
<thead>
<tr>
<th>Design solution meets:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allocated performance requirements</td>
</tr>
<tr>
<td>Functional performance requirements</td>
</tr>
<tr>
<td>Interface requirements</td>
</tr>
<tr>
<td>Workload limitations</td>
</tr>
<tr>
<td>Constraints</td>
</tr>
<tr>
<td>Use models and/or prototypes to determine success</td>
</tr>
</tbody>
</table>
### IEEE 1220, (6.6): Success Criteria (continued)

<table>
<thead>
<tr>
<th>Design solution satisfies</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>Functional architecture</em></td>
</tr>
<tr>
<td><em>Requirements baseline</em></td>
</tr>
<tr>
<td>(Use models and/or prototypes)</td>
</tr>
</tbody>
</table>

- *Requirements* of the lowest level of the design architecture, including derived requirements, are *traceable* to the verified functional architecture.
# Design Solution

## Work Products

<table>
<thead>
<tr>
<th>IEEE 1220, (6.5, 6.6): Work Products</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Integrated data package to document the selected design elements:</td>
</tr>
<tr>
<td>– Drawings</td>
</tr>
<tr>
<td>– Schematics</td>
</tr>
<tr>
<td>– Software documentation</td>
</tr>
<tr>
<td>– Manuals</td>
</tr>
<tr>
<td>– Procedures</td>
</tr>
</tbody>
</table>
### Design Solution Work Products

<table>
<thead>
<tr>
<th>IEEE 1220, (6.5, 6.6): Work Products</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Design solution alternatives</td>
</tr>
<tr>
<td>• Physical interfaces</td>
</tr>
<tr>
<td>• Models and prototypes</td>
</tr>
<tr>
<td>• Failure modes and effects analyses (FMEA)</td>
</tr>
<tr>
<td>• Requirements traceability and allocation matrices</td>
</tr>
<tr>
<td>• Trade off analysis results</td>
</tr>
<tr>
<td>• Finalized design and description of interfaces</td>
</tr>
</tbody>
</table>
EVM Process Improvement through Capability Maturity Model Integration® (CMMI)
Product Requirements

• CMMI, PMBOK Guide: Traceability and consistency

Requirements

Product Requirements
Baseline

Work

• Project Plans
  Task 1
  Task 2
  Task 3
• Activities
• Work Products
• CMMI Specific Practice (SP) 1.1:
  • Establish *measurement objectives* derived from *information needs* and *objectives*

• CMMI SP 1.2:
  • Specify *quantifiable* measures to address measurement objectives
    • Stated in *precise, unambiguous* terms
    • *Operational definitions* for the measures
    • Specify *how measurement data* will be obtained

• EVMS: “by *management assessment*”
Process and Product QA

• Product QA
  • CMMI:
    • *Objectively* evaluate work products against *clearly stated criteria*
    • *Minimize subjectivity*
  • EVMS:
    • EV is measurement of *quantity* of work
    • “*Quality and technical* content of work performed are *controlled by other means!*”
CMMI Typical Work Products

• Requirements Development
  • Product and product-component requirements
  • Interface requirements
  • Required functionality
  • Product component operational concepts, scenarios and environments
  • TPMs
CMMI Typical Work Products

- Technical Solution
  - Operational concepts and scenarios
  - Technical data package
    - Allocated requirements
    - Product-component descriptions
    - Key product characteristics
    - Required physical characteristics and constraints
  - Interface requirements
  - Material requirements
  - Verification criteria to achieve requirements
CMMI Typical Work Products

• Technical Solution
  • Comprehensive product-component interface
    • Interface design specs.
    • Interface control documents
    • Interface specification criteria
  • Implemented design
CMMI Typical Work Products

- Requirements management
  - Requirements traceability matrix (RTM)
- Verification
  - Exit and entry criteria for work products
  - Verification results
- Measurement and analysis
  - Specifications of base and derived measures
- Decision analysis and resolution
  - Results of evaluating alternative solutions (trade-studies)
PMBOK® Guide

PMBOK®, Guide to Project Management

© 2006 Paul J. Solomon
PMBOK® Guide (5.5).

- Product scope
  - Features and functions that characterize a
    - Product
    - Service
    - Result
- Project scope
  - Work that needs to be accomplished to deliver a
    - Product, service or result
      with the specified features and functions.
• Establish a quality baseline as part of the PMB (8.1.3.5)
  – Integrate technical and quality objectives (10.3.1.5)
• Specify TPMs to measure schedule performance (11.6.2.4)
4. Project Management with Performance-Based Earned Value® (PBEVSM)
4 Principles and 16 Guidelines

Specify most effective measures of project performance

Requirements-driven plan

Consistent with standards and models

Tailorable and scalable, depending on risk

Lean
PBEV and Quality

• Link EV with evolving development maturity or quality

• **Quantify** quality measures
  – Percent of product requirements met (weighted)
  – Technical performance achieved

• **Measure quality**
  – Of “completed” work products
  – Of work in process

*EV without Quality has less management value*
PBEV Based on Standards and Models

- ANSI/EIA-632
- IEEE 1220
- CMMI®
- PMBOK® Guide
- INCOSE SE Handbook
- PSM. Practical Software and Systems Measurement: A Foundation for Objective Project Management
• Integrates SE with EVM
  – Planning:
    • Link performance measurement baseline (PMB) to:
      – Product requirements (technical) baseline
      – SEP
      – SE process work products
    • Identify product metrics for performance-based progress measurement
      – Planned value profile of TPMs
      – Planned development maturity to date
  • Success criteria (reviews and work products)
PBEV Characteristics

- Integrates SE with EVM
  - Measurement
    - Objective measurement of interim progress
    - Progress of requirements through engineering life cycle
    - EV linked with
      - Indicated quality of end product
      - TPM achieved
  - EV used to measure Quality
    - Not just work accomplished
• Meaningful analysis
  – Correlate analyses of deviations from plan:
    • Technical maturity/quality
    • Schedule
    • Cost
PBEV Characteristics

• Lean
  – Minimizes costs; measurement costs money
  – Fewer work packages with right base measures
    • Product requirements-driven
    • Quality measures
    • Work products
• Applicable to all development models and methods
  – Waterfall, incremental, spiral, V, evolutionary, agile
Principles of PBEV

1. Integrate product requirements and quality into the project plan.
2. Specify performance towards meeting product requirements, including planned quality, as a base measure of earned value.
3. Integrate risk management with EVM.
4. Tailor the application of PBEV according to the risk.
Supplemental PBEV Process Flow

(P) Establish product requirements and components (technical baseline)

Guideline 1.1

(P) Integrate product requirements and quality with plan

Guidelines 1.2, 2.2

(P) Integrate risk management with plan

Guidelines 3.2, 3.2, 4.1, 4.2

(P) Measure progress towards meeting product requirements and quality

Guideline 2.7

Define the work (WBS)

Plan the work (Schedule & Budget)

Execute the plan

Measure the work

Analyze variances

Incorporate internal/external changes

Implement corrective action

(P) = Supplemental PBEV Process

© 2006 Paul J. Solomon
1.1 Establish *product requirements* and allocate these to product components.

1.2 Maintain *bidirectional traceability* of *product* and product component *requirements among*:

- Project plans
- Work packages and planning packages
- Work products.
1.3 Identify *changes* that need to be made to
- Project plans
- Work packages
- Planning packages
- Work products *resulting from changes to the product requirements.*

2.1 *Define the information need and objective to measure progress towards satisfying product requirements.*
2.2 Specify *work products* and performance-based *measures* of progress for meeting *product requirements* as *base measures of earned value*.
2.3 Specify *operational definitions* for the base measures of EV,

stated in *precise, unambiguous terms*

Address:

- Communication
  - What has been measured
  - How it was measured
  - What are the units of measure
  - What has been included or excluded

Repeatability: can the measurement be repeated, given the same definition, to get the same results?
2.4 Identify *event-based success criteria* for technical reviews:

- *Development maturity to date*
- Product’s ability to meet *product requirements.*
2.5 Establish:
• Time-phased, \textit{planned values} for measures of \textit{progress towards meeting product requirements}
• Dates or frequency for checking progress
• Dates when \textit{full conformance will be met}.

2.6 Allocate budget in discrete work packages to measures of progress towards meeting \textit{product requirements}.
2.7 Compare

- Amount of planned budget and
- Amount of budget earned

for achieving progress towards meeting *product requirements*
2.8 Use the level of effort (LOE) method to plan work that is measurable but is not a measure of progress towards meeting

- Product requirements
- Final cost objectives
- Final schedule objectives.

2.9 Perform more effective variance analysis by segregating discrete effort from LOE.
3.1 Identify *changes* that need to be made to
- Project plans
- Work packages
- Planning packages
- Work products
  resulting from *responses to risks*.

3.2 Develop revised EAC
based on *risk quantification*
4.1 Tailor the application of PBEV to the elements of the WBS according to the risk.

4.2 Tailor the application of PBEV to the phases of the system development life cycle according to the risk.
Requirements Development and Management
Manage Requirements

• Second most critical requirements practice
• Example: Use an additional radio band width.
• Changes to plan
  – Trade studies to determine best solution.
  – Budget and schedule changed.
  – All subsequent milestones moved to right
  – Higher cost to customer caused by
    • Level of effort activities extended
    • Skill retention (of people on discrete tasks).
Trade Studies

- Provide objective foundation to select an approach to the solution of an engineering problem.
- Typical trade results:
  - Select user/operational concept
  - Select system architectures
  - Derive requirements
    - Alternative functional approaches to meet requirements
  - Requirements allocations
  - Technical/design solutions
  - Cost analysis results
  - Risk analysis results
Maintaining the Technical Requirements Baseline

• Baseline
  – Specification or product that has been formally reviewed and agreed on.
  – Serves as the basis for further development.
  – Changed only through formal change control procedures.

• Maintaining *product* requirements baseline supports planning and control.
DOD Guidance on SE

Good SE planning:
• Manage the technical baselines
• Technical baselines:
  • Are specific SE work products
  • Provide product-driven view for SE cost management
• Maturity of baselines are entry criteria for event-based technical reviews
• EV provides critical insight to technical progress

Source: Office of the Undersecretary of Defense (Acquisition, Technology & Logistics)/Defense Systems website
Risk Management
Risk vs. Issue

Source: Risk Management Guide for DOD Acquisition:

• If root cause is described in past tense, it has already occurred. It is an issue.
• Incorporate risk mitigation activities into the IMS and EVM
  – Monitor progress against risk plans
• Include quantified risk impacts in EAC
## EVMS: Not a Risk Management Tool

<table>
<thead>
<tr>
<th>Significant Variance</th>
<th>Issue?</th>
<th>Risk?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cost</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>Schedule</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

© 2006 Paul J. Solomon
Implementing PBEV into Your Project
PBEV Techniques

• Allocate budget to completion of
  – Enabling work products (drawings, code)
  – Allocated requirements
• Establish milestones with success criteria
  – SE Life Cycle Work Products
  – Number of allocated requirements to be met
  – Technical performance
    • Planned technical maturity
    • Quality
PBEV Techniques

- Measure quality
  - Work products (partial and complete)
  - Technical maturity of evolving product
  - Use analysis, models, simulations, prototypes
- Base EV on
  - Work products (drawings, code) and
  - Quality
PBEV Techniques

• Use LOE if work is measurable but is *not* a measure of progress towards meeting:
  – Product requirements
  – Final cost objectives
  – Final schedule objectives
Initial Design Development Measures

- Design (work unit progress):
  - Base EV on
    - # Enabling work products and # Requirements met
  - Example:
    - # Components designs completed and
    - # Requirements met traced to components

- Recommended PBEV Measure
EX 1: EV Based on Drawings and Requirements

• SOW: Design a subsystem with 2 TPM requirements:
  – Maximum (Max.) weight: 200 lb.
  – Max. diameter: 1 inch
• Enabling work products: 50 drawings
• BAC: 2000 hours
  – Drawings: 40 hours/drawing @ 50
  – Requirements *not* met on schedule:
    • Potential negative EV
      – Weight: -100
      – Diameter -200
**EX 1: Schedule Plan and Status**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Drawings</td>
<td>8</td>
<td>10</td>
<td>12</td>
<td>10</td>
<td>10</td>
<td>50</td>
</tr>
<tr>
<td>Requirements met:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Weight</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Diameter</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Status at April 30

- Drawings completed: 41
- Weight requirement *not* met
- Diameter requirement met
## EX 1: Earned Value

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Planned drawings cur</td>
<td>8</td>
<td>10</td>
<td>12</td>
<td>10</td>
<td>10</td>
<td>50</td>
</tr>
<tr>
<td>Planned drawings cum</td>
<td>8</td>
<td>18</td>
<td>30</td>
<td>40</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>BCWS cur</td>
<td>320</td>
<td>400</td>
<td>480</td>
<td>400</td>
<td>400</td>
<td>2000</td>
</tr>
<tr>
<td>BCWS cum</td>
<td>320</td>
<td>720</td>
<td>1200</td>
<td>1600</td>
<td>2000</td>
<td>2000</td>
</tr>
<tr>
<td>Actual drawings completed cur</td>
<td>9</td>
<td>10</td>
<td>10</td>
<td>12</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>Actual drawings completed cum</td>
<td>9</td>
<td>19</td>
<td>29</td>
<td>41</td>
<td>49</td>
<td></td>
</tr>
<tr>
<td>EV (drawings) cum</td>
<td>360</td>
<td>760</td>
<td>1160</td>
<td>1640</td>
<td>1960</td>
<td></td>
</tr>
<tr>
<td>Negative EV Reqs cum</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>-100</td>
<td></td>
</tr>
<tr>
<td>Net EV cum</td>
<td>360</td>
<td>760</td>
<td>1160</td>
<td>1640</td>
<td>-1860</td>
<td></td>
</tr>
</tbody>
</table>

SV = -140
EX 1: Variance Analysis

Variance analysis (drawings and requirements):

- 1 drawing behind schedule - 40
- Diameter requirement met - 0
- Weight requirement *not* met: - 100

Schedule variance - 140
TPM at Higher WBS Level

• Design of a component at the work package level
• Completion of the comp. design depends on
  – Achieving allocated TPMs values at
    1. Component level and
    2. Subsystem level
• EV is dependent on planned TPM values achieved at both levels
TPM at Higher WBS Level

• For a weight TPM, all components play a part
• For other TPMs, such as response time
  – Subsets of the components combine to meet subsystem performance objectives
    • Hardware components
    • Software components
TPM at Higher Level

• Assumptions:
  – Component in Example 3 is one of four components that form a subsystem
  – Subsystem’s TPM objective is 4000 lb.
  – SEP states:
    Some components may be overweight at completion if there are offsets in other components (Comp)
    as long as the total subsystem (Sub) weight does not exceed 4000 lb.
## EX 2: TPM at Higher WBS Level

<table>
<thead>
<tr>
<th>Component, Work Pkg.</th>
<th>TPM Planned Value</th>
<th>Planned Completion</th>
<th>Component EV Penalty</th>
<th>Subsystem EV Penalty</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>200</td>
<td>April</td>
<td>-100</td>
<td>-50</td>
</tr>
<tr>
<td>2</td>
<td>1000</td>
<td>April</td>
<td>-500</td>
<td>-250</td>
</tr>
<tr>
<td>3</td>
<td>2000</td>
<td>May</td>
<td>-1000</td>
<td>-500</td>
</tr>
<tr>
<td>4</td>
<td>800</td>
<td>May</td>
<td>-400</td>
<td>-200</td>
</tr>
<tr>
<td><strong>Subsystem total</strong></td>
<td><strong>4000</strong></td>
<td></td>
<td><strong>-2000</strong></td>
<td><strong>-1000</strong></td>
</tr>
</tbody>
</table>
## TPM at Higher WBS Level

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Planned drawings</td>
<td>8</td>
<td>10</td>
<td>12</td>
<td>10</td>
<td>10</td>
<td>50</td>
</tr>
<tr>
<td>cur</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Planned drawings</td>
<td>8</td>
<td>18</td>
<td>30</td>
<td>40</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>cum</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BCWS cur</td>
<td>320</td>
<td>400</td>
<td>480</td>
<td>400</td>
<td>400</td>
<td>2000</td>
</tr>
<tr>
<td>BCWS cum</td>
<td>320</td>
<td>720</td>
<td>1200</td>
<td>1600</td>
<td>2000</td>
<td>2000</td>
</tr>
<tr>
<td>Actual drawings</td>
<td>9</td>
<td>10</td>
<td>10</td>
<td>12</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>completed cur</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Actual drawings</td>
<td>9</td>
<td>19</td>
<td>29</td>
<td>41</td>
<td>49</td>
<td></td>
</tr>
<tr>
<td>completed cum</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EV (drawings)</td>
<td>360</td>
<td>760</td>
<td>1160</td>
<td>1640</td>
<td>1960</td>
<td></td>
</tr>
<tr>
<td>cum</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Negative EV Reqs</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>-1500</td>
<td></td>
</tr>
<tr>
<td>cum</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Net EV cum</td>
<td>360</td>
<td>760</td>
<td>1160</td>
<td>1640</td>
<td>460</td>
<td></td>
</tr>
</tbody>
</table>
## Ex. 3: Rework in Same WP

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Planned drawings –cur.</td>
<td>8</td>
<td>10</td>
<td>12</td>
<td>10</td>
<td>10</td>
<td>50</td>
</tr>
<tr>
<td>Planned drawings –cum.</td>
<td>8</td>
<td>18</td>
<td>30</td>
<td>40</td>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td>BCWS – cum.</td>
<td>320</td>
<td>720</td>
<td>1200</td>
<td>1600</td>
<td>2000</td>
<td>2000</td>
</tr>
<tr>
<td>Drawings completed</td>
<td>9</td>
<td>10</td>
<td>10</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Drawings returned</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>-5</td>
<td></td>
</tr>
<tr>
<td>Net drawings – cur.</td>
<td>9</td>
<td>10</td>
<td>10</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Net drawings – cum.</td>
<td>9</td>
<td>19</td>
<td>29</td>
<td>28</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Net EV – cur.</td>
<td>360</td>
<td>400</td>
<td>400</td>
<td>-40</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EV – cum.</td>
<td>360</td>
<td>760</td>
<td>1160</td>
<td>1120</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SV – cum.</td>
<td>0</td>
<td>40</td>
<td>-40</td>
<td>-480</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Drawings Returned for Rework Result in Negative EV**
Ex 4: Rework in Separate WP

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>WP 1: Initial development of drawings</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Planned number of drawings</td>
<td>8</td>
<td>10</td>
<td>12</td>
<td>10</td>
<td>10</td>
<td>50</td>
</tr>
<tr>
<td>BCWS</td>
<td>288</td>
<td>360</td>
<td>432</td>
<td>360</td>
<td>360</td>
<td>1800</td>
</tr>
</tbody>
</table>

Planned rework in WP 2

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>BCWS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>200</td>
</tr>
</tbody>
</table>
Ex 4: Rework in Separate WP

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>MS 1</strong></td>
<td><strong>MS 2</strong></td>
<td><strong>MS 3</strong></td>
<td></td>
</tr>
<tr>
<td>BCWS</td>
<td></td>
<td></td>
<td>60</td>
<td>70</td>
<td>70</td>
<td>200</td>
</tr>
</tbody>
</table>

Rework Milestones:

- Milestone 3: 100% of drawings meet requirements
- Milestone 2: 90% of drawings meet requirements
- Milestone 1: 80% of drawings meet requirements
EX 5: Trade Study

• Outcome is usually a recommendation that is needed to make a decision.
• Decision constrains and guides further progress.
• Work product: documented trade study results.
• Engineering processes should include a process and structured approach for performing trade studies.
  – Process should include both interim and final work products that can be:
    • Planned, scheduled, and measured.
Trade Study Outline

1. Purpose of Study:
   – Resolve an issue
   – Perform decision analysis
   – Perform analysis of alternatives

2. Scope of study
   – State level of detail of study
   – State assumptions
   – Identify influencing requirements and constraints.
3. Trade study description

Describe trade studies to be performed to make tradeoffs among:

– Concepts
– User requirements
– System architectures
– Design
– Program schedule
– Functional performance requirements
– Life-cycle costs
4. Analytical approach
   – Identify candidate solutions
   – Measure performance
   – Develop models and measures of merit
   – Develop values for viable candidates
   – Selection criteria: (normally risk, performance, cost)
5. Scoring
   – Determine measures of results to be compared to criteria
   – Assign weights to measures of results reflecting their relative importance
   – Perform sensitivity analysis

6. Evaluate alternatives

7. Documentation of trade results
## Trade Study Schedule

### Trade Study Base Measures: Evaluate Alternatives

<table>
<thead>
<tr>
<th>Initial evaluation of each of 5 candidates has three milestones:</th>
<th>Time Period</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Start test set up</td>
<td>1</td>
</tr>
<tr>
<td>• Tests executed to completion</td>
<td>2</td>
</tr>
<tr>
<td>• Analyze and document</td>
<td>3</td>
</tr>
</tbody>
</table>

| Down select from 5 candidates to 2 candidates                 | 3           |
| Document recommendation                                      | 4           |
Trade Study EV

• Evaluation activity planning assumptions

• Total Budget: 1000 hours
  • Test and evaluate 5 candidates: 500
    – 100 per candidate
    – Take EV even if candidate discarded before test complete
  • Down select to 2 candidates: 200
  • Document final recommendation: 300

• Period of Performance: 4 months
EX 6: Requirements Management

- Discretely measure requirements management
- Use RTM to control plan
- Requirements management (RM) tasks
  - Defined
  - Validated
  - Determined verification method
  - Approved
  - Allocated
  - Traced to verification document (test procedure)
  - Tested
  - Verified
- Key indicator of project performance
– # requirements traced to software or hardware components

Note: Budget per Work Unit does not have to be equally distributed

- Recommended PBEV Measure
# Budget Allocation

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Budget %</strong></td>
<td></td>
<td></td>
<td></td>
<td>15%</td>
<td>15%</td>
<td>20%</td>
<td>15%</td>
<td>20%</td>
<td>100%</td>
</tr>
<tr>
<td><strong>Component</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Enclosure</td>
<td>3</td>
<td>240</td>
<td>36</td>
<td>36</td>
<td>36</td>
<td>48</td>
<td>36</td>
<td>48</td>
<td>240</td>
</tr>
<tr>
<td>Transmitter</td>
<td>1</td>
<td>80</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>16</td>
<td>12</td>
<td>16</td>
<td>80</td>
</tr>
<tr>
<td>Battery</td>
<td>2</td>
<td>160</td>
<td>24</td>
<td>24</td>
<td>24</td>
<td>32</td>
<td>24</td>
<td>32</td>
<td>160</td>
</tr>
<tr>
<td>Control</td>
<td>1</td>
<td>80</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>16</td>
<td>12</td>
<td>16</td>
<td>80</td>
</tr>
<tr>
<td>Software</td>
<td>9</td>
<td>720</td>
<td>108</td>
<td>108</td>
<td>108</td>
<td>144</td>
<td>108</td>
<td>144</td>
<td>720</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>16</td>
<td>1280</td>
<td>192</td>
<td>192</td>
<td>192</td>
<td>256</td>
<td>192</td>
<td>256</td>
<td>1280</td>
</tr>
</tbody>
</table>
# Time-Phased Budget

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Defined</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Validated</td>
<td></td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Verif. Method</td>
<td></td>
<td></td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Allocated</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Traced to Verif.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Verified</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>BCWS current</th>
<th>Budget/Activity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Defined</td>
<td>12</td>
</tr>
<tr>
<td>Validated</td>
<td>12</td>
</tr>
<tr>
<td>Verif. Method</td>
<td>12</td>
</tr>
<tr>
<td>Allocated</td>
<td>16</td>
</tr>
<tr>
<td>Traced to Verif.</td>
<td>12</td>
</tr>
<tr>
<td>Verified</td>
<td>16</td>
</tr>
</tbody>
</table>

| Total              | 36              | 24              | 24              | 24              | 48              | 36              | 48              | 240               |

| BCWS cumulative    | 36              | 60              | 84              | 108             | 156             | 192             | 240             |
## Earned Value

<table>
<thead>
<tr>
<th>Enclosure</th>
<th>Jan</th>
<th>Feb</th>
<th>Mar</th>
<th>Apr</th>
<th>May</th>
</tr>
</thead>
<tbody>
<tr>
<td>Completed</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Budget/Activity</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Defined</td>
<td>12</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Validated</td>
<td>12</td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Verif. Method</td>
<td>12</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EV cumulative</td>
<td>0</td>
<td>36</td>
<td>36</td>
<td>60</td>
<td>72</td>
</tr>
<tr>
<td>BCWS cumulative</td>
<td>36</td>
<td>60</td>
<td>84</td>
<td>108</td>
<td>156</td>
</tr>
<tr>
<td>Schedule Variance</td>
<td>-36</td>
<td>-24</td>
<td>-48</td>
<td>-48</td>
<td>-84</td>
</tr>
</tbody>
</table>
EX 7: PBEV Variance Analysis

• Negative EV causes sudden schedule and cost variances
• Example: New schedule variance when TPM planned value not achieved.
  – Requirements for control console:
    • Maximum surrounding temperature (Max.) < 100 degrees F. for more than 30 seconds
    • Max. never > 120 degrees
  – Prior status
    • Designs on schedule
      – Control console
      – Nearby equipment
      – Cooling methods
    • Thermal analyses on schedule
    • Meets TPM planned values at lower WBS levels
PBEV Variance Analysis

- Known performance issue
  - TPM planned value not achieved
  - Max. > 120 degrees
- Negative EV results in significant schedule variance
- How to describe in variance analysis?
PBEV Variance Analysis

• Cause:
  – Insufficient space between surrounding components
  – Insufficient airflow to cool the equipment
  – Root cause:
    • Requirements did not limit dimensions of cables and connectors
• Impact:
  – 4 week delay for redesign
  – Cost increase of & 50 K for redesign, retest
• Corrective Action Plan:
  – Rework requirements, design, test
  – Improve requirements development and validation process
PBEV EAC Tip

- If significant technical issues exist, only detailed planning can provide reliable EAC
- If significant risks exist (high probability and cost impact), include cost impact in EAC
IT/ Software Progress Measurement Issues
Initial Development: Incremental Capability

- Document baseline content of incremental builds
  - # functional requirements
  - # components
- Baseline the build milestones and completion criteria
- Baseline the build work packages and EV metrics
- Take EV based on functionality achieved
  - Show completed milestones & take full earned value
    Only if all completion criteria and planned functionality attained
Internal Replanning of Deferred Functionality

- If build is released short of planned functionality:
  - Take *partial* EV and leave work package open
  - Take *partial* EV and close work package

- Transfer deferred scope and budget to first month of work package for next incremental build
  - EV mirrors technical performance
  - Schedule variance retained

- Disclose shortfall and slips on higher schedules
EX 8: Deferred Functionality

- SOW: Software Requirements in 2 Builds:

<table>
<thead>
<tr>
<th>Build</th>
<th>Allocated Req.</th>
<th>Budget/Req.</th>
<th>BAC</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>100</td>
<td>5</td>
<td>500</td>
</tr>
<tr>
<td>B</td>
<td>60</td>
<td>5</td>
<td>300</td>
</tr>
</tbody>
</table>
## SW Build Plan

<table>
<thead>
<tr>
<th>Build A</th>
<th>Jan</th>
<th>Feb</th>
<th>Mar</th>
<th>Apr</th>
<th>May</th>
<th>Jun</th>
<th>Jul</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Planned Reqs. met</td>
<td>25</td>
<td>25</td>
<td>25</td>
<td>25</td>
<td></td>
<td></td>
<td></td>
<td>100</td>
</tr>
<tr>
<td>Budget/Req.: 5 hours</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BCWS current (cur)</td>
<td>125</td>
<td>125</td>
<td>125</td>
<td>125</td>
<td></td>
<td></td>
<td></td>
<td>500</td>
</tr>
<tr>
<td>BCWS cumulative (cum)</td>
<td>125</td>
<td>250</td>
<td>375</td>
<td>500</td>
<td></td>
<td></td>
<td></td>
<td>500</td>
</tr>
</tbody>
</table>

| Build B                                      |     |     |     |     |     |     |     |       |
| Planned Reqs. Met                            |     |     |     |     | 20  | 20  | 20  | 60    |
| BCWS cur                                    | 100 | 100 | 100 | 300 |     |     |     |       |
# Deferred Functionality Status

<table>
<thead>
<tr>
<th>Build A</th>
<th>Jan</th>
<th>Feb</th>
<th>Mar</th>
<th>Apr</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Planned Reqs. Met cur</td>
<td>25</td>
<td>25</td>
<td>25</td>
<td>25</td>
<td>100</td>
</tr>
<tr>
<td>Actual Reqs. Met cur</td>
<td>20</td>
<td>20</td>
<td>25</td>
<td>25</td>
<td>90</td>
</tr>
<tr>
<td>BCWS cur</td>
<td>125</td>
<td>125</td>
<td>125</td>
<td>125</td>
<td>500</td>
</tr>
<tr>
<td>EV cur</td>
<td>100</td>
<td>100</td>
<td>125</td>
<td>125</td>
<td>450</td>
</tr>
</tbody>
</table>

| BCWS cum                | 125 | 250 | 375 | 500 |
| EV cum                  | 100 | 200 | 325 | 450 |

**Schedule variance (SV):**

| Reqs. Met | -5 | -10 | -10 | -10 |
| SV        | -25| -50 | -50 | -50 |
### Deferred Functionality Replan

<table>
<thead>
<tr>
<th></th>
<th>Apr</th>
<th>May</th>
<th>Jun</th>
<th>Jul</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Close Build A work package</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SV:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reqs. Not Met</td>
<td>-10</td>
<td></td>
<td></td>
<td>-10</td>
<td></td>
</tr>
<tr>
<td>BCWS remaining</td>
<td>-50</td>
<td></td>
<td></td>
<td>-50</td>
<td></td>
</tr>
<tr>
<td><strong>Build B</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Before Replan:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Planned Reqs. Met</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BCWS cur</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Plus transfer:</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reqs.</td>
<td>+10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BCWS</td>
<td>+50</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>After replan:</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Planned Reqs. Met</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BCWS cur</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

© 2006 Paul J. Solomon

120
# Deferred Functionality Status

<table>
<thead>
<tr>
<th>Build B After Replan:</th>
<th>May</th>
<th>Jun</th>
<th>Jul</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Planned Reqs. Met</td>
<td>30</td>
<td>20</td>
<td>20</td>
<td>70</td>
</tr>
<tr>
<td>BCWS cur</td>
<td>150</td>
<td>100</td>
<td>100</td>
<td>350</td>
</tr>
<tr>
<td>Actual Reqs. Met cur</td>
<td>20</td>
<td></td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>EV cur</td>
<td>100</td>
<td></td>
<td>100</td>
<td></td>
</tr>
</tbody>
</table>

Schedule variance cum:

| Reqs. Met                          | -10 |
| SV                                 | -50 |
Software Quality Measures

- Software quality measures are TPMs
  - Defect density
  - Number of problem reports
- Failure to achieve planned quality indicates
  - More rework during development
  - More problems after product delivery
Ex 9: Quality as a Limiter of EV

- TPM: Number or density of defects
- If defect plan/quality goals are not attained:
  Use Hammervold Algorithm:
  - Assumes that more defects than planned will be found in future
  - Assumes that some verified requirements will be negated by future defects
  - Limits EV to 80% of BAC even if % requirements met > 80%
Implementing Better Acquisition Management into Your Project
Acquisition Management

Ensure Contractors Integrate SE with EVM

- Requirements, incentives, insight:
  - Solicitation/Request for Proposal (RFP)
  - Integrated Master Plan (IMP)
  - Integrated Baseline Review (IBR)
  - Integrated Master Schedule (IMS)
  - EVMS compliance assessments
  - Independent technical assessments
  - Monitor consistency and validity of reports
  - Independent EAC and risk assessments
  - Award fee criteria
• Event-based technical reviews
  – Appropriate entry and exit criteria
• Show build up from unit design and test to subsystem to system integration
• Software development and integration approach reflected in significant accomplishments and criteria
• Review implementation of SE:
  – Entry and success criteria for IMP events
  – Requirements management and traceability
  – Control points for product metrics and TPMs
  – Milestones with technical maturity success criteria
    • TPM planned values
    • Meeting requirements
    • Percent of designs complete
  – SE life cycle work products in IMS
• Confirm integration of
  • Technical baseline
  • WBS
  • IMP/IMS
  • EVM
  • Risk management
IMP > IMS Flowdown

- Meeting all the IMP criteria indicates completion of the significant accomplishment.
- The IMS should decompose IMP criteria into tasks necessary to meet the criteria.
IMS Checklist

- Review completion criteria for significant accomplishments
  - Objective and measurable
- Review each significant accomplishment
  - Do events occur at the system level and across multiple Integrated Product Teams (IPT)?
  - Does each criterion relate to a specific IPT?
IMS Checklist

• Subcontracted efforts
  • Appropriate visibility?
    • Requirements flowdown
    • Design reviews prior to system level reviews
  • IMS traceable to EVMS work and planning packages
    • Timing
    • Completion criteria
    • WBS numbers
Independent Technical Assessments

- Verify technical maturity
  - Product and Quality metrics
    - TPM achievement
    - Requirements met
Monitor Consistency and Validity of Reports

- Compare performance reports for consistency:
  - Program status
    - Technical
    - Schedule
    - EV
  - Variance analyses
    - Root causes
    - Corrective action plans
    - Impacts on cost and schedule
Independent Assessments of EAC and Risks

• Perform EAC and quantified risk assessments
  – At total contract level
  – At lower WBS levels
• Compare your assessment with supplier’s
• Resolve significant differences
• Validate supplier’s corrective action plans
  – Performance efficiency
  – Schedule recovery
Award Fee Criteria for Successful I BR

• Agree on:
  – Entry and exit criteria for event-based technical reviews
  – Objective completion criteria for each significant accomplishment needed to support reviews.

• Subcontracted efforts on IMS have sufficient:
  – Milestones
  – Accomplishments
  – Completion criteria

• Completions of subcontractor design reviews occur prior to system level design reviews.
Award Fee Criteria for Successful IBR

- Management process provides effective:
  - Integrated technical/schedule/cost planning
  - Baseline control
- Valid critical path
- Technical baselines are included in the IMS
- TPM milestones are in IMS with planned values
- IMS milestones and completion criteria traceable to EVMS work and planning packages
Award Fee Criteria for Successful I BR

• Process, bidirectional traceability among:
  – Requirements
  – Work products
  – Project plans (IMS, work and planning packages)
• IMS includes activities identified in risk mitigation
• PMB is sufficient to successfully execute the project
Award Fee Criteria for Successful Technical Reviews

• All exit criteria for event-driven technical reviews met on schedule
  – Development maturity is on schedule
  – Issues resolved
    • All subsystems
    • Products
    • Life cycle processes
• Unacceptable risks are mitigated
Award Fee Criteria for Successful Technical Reviews

- System design is capable of meeting requirements
- Cost performance objectives have been met
- Bidirectional traceability is maintained among the requirements and the project plans, work products, and work packages.
- Accomplishments and plans satisfy criteria for continuation of the technical effort
Summary

• Integrate
  – Systems engineering with PBEV
    • Product requirements
    • Manage the technical baseline
    • Technical performance measures
    • SE life cycle work products
    – Technical>schedule>cost performance
• Lean process
  – Less work packages with right base measures
• Agile
Benefits of Using PBEV

- Management Decisions
- Identify Problem Areas
- Take Corrective Action
- Change Future Direction
- Forecast Future Performance

EVMS Input
- Budget
- Schedule
- EV
- Actual Costs

Risk Management Output

EVMS Output
- Technical Requirements / Quality

Data Analysis
Process Improvement

Using CMMI to Improve Earned Value Management

Dave Burgess
Cost Department Head

Ted Rogers
EVM Division Head

Chris Mushrush
EVM Subject Matter Expert

Dave Kester
EVM Subject Matter Expert

October 2004

Points of Contact
Process: Earned Value Management AIR 4.2.3
Technical: Software Engineering AIR 4.1.4

CrossTalk: The Journal of Defense Software Engineering

Sept. 2001
Aug. 2005
May 2006

SEI / CMMI

NAVAIR

Avionics Modernization

Carpe diem

Unlimited distribution subject to the copyright.
But wait.  
There’s more!  
• Examples  
• Templates  
• Tips  
• Standards  
• FAR
Questions?
References

- CMMI is registered by Carnegie Mellon University in the U.S. Patent and Trademark Office.
- Performance-Based Earned Value is registered by Paul Solomon in the U.S. Patent and Trademark Office. PBEV is a Service Mark of Paul Solomon.
- PMBOK is registered by the Project Management Institute in the U.S. Patent and Trademark Office.

© 2006 Paul J. Solomon
References


• Young, Ralph R. *Effective Requirements Practices* (Addison-Wesley, 2001).


Acronyms

- BCWP: Budgeted Cost for Work Performed
- BCWS: Budgeted Cost for Work Scheduled
- EV: Earned Value = BCWP
- EVMS: Earned Value Management System
- IBR: Integrated Baseline Review
- IMP: Integrated Master Plan
- IMS: Integrated Master Schedule
- PBEV: Performance-Based Earned Value
- SEP: Systems Engineering Plan
- SOW: Statement of Work
- TPM: Technical Performance Measure
- WBS: Work Breakdown Structure
9TH ANNUAL SYSTEMS ENGINEERING CONFERENCE

“Focusing on Improving Performance of Defense Systems Programs”

Event #7870
October 23-26, 2006
San Diego, CA

Sponsored by the National Defense Industrial Association, with Technical Co-Sponsorship by IEEE AES, IEEE Systems Council and INCOSE

Supported by the Office of Under Secretary of Defense for Acquisition, Technology and Logistics, Director, Systems and Software Engineering and Office of the DoD Chief Information Officer
A major Conference focusing on improving acquisition and performance of Defense programs and systems, including net-centric operations and data/information interoperability, system-of-systems engineering, and all aspects of system sustainment, will be convened in San Diego, CA, October 23-26, 2006. This Conference is sponsored by the National Defense Industrial Association, Systems Engineering Division, with technical co-sponsorship by IEEE AES, IEEE Systems Council, and the International Council on Systems Engineering, and is supported by the Office of Under Secretary of Defense for Acquisition, Technology and Logistics, Director, Systems and Software Engineering and Office of the DoD Chief Information Officer.

Background
The Department of Defense in undertaking a major transformation of our military capability in response to the new world environment and unforeseen threats. The ability to effect this transformation can only be realized if our Defense systems - space, air, land, sea, and under sea - can effectively satisfy mission area and capability requirements and achieve and sustain a high degree of interoperability, systems integration, systems safety, readiness, and availability. We believe that the greatest opportunity to achieve these objectives for new and legacy systems is through strong technical management embodied in systems engineering methodologies and processes, on the part of both industry and the DoD. Strong emphasis on systems engineering across the full acquisition life cycle, from concept refinement through sustainment, is a key enabler of incremental development and evolutionary acquisition. The Systems Engineering Conference is an annual event targeted at exploring the role of technical planning and execution in Defense programs and systems from a variety of perspectives, academic and pragmatic, by the entire Defense systems engineering community.

Conference Objectives
This conference seeks to create an interactive forum for Program Managers, Systems Engineers, Chief Scientists, and engineers and managers from the government, industry, and academic communities whose interests converge on Defense acquisition, from capabilities analysis through disposal. This Conference will provide the opportunity to shape policy and guidance through the exchange of innovative procedures and lessons learned to address the following current issues:

- Interoperability & Systems Integration
- System-of-Systems Engineering
- Life Cycle Systems and Program Management
- Aging Aircraft
- Life Cycles Systems Management
- Sustainment & Upgrade of Legacy Systems
- Application of Government & Industry "Best Practices" Yools, Methodologies & Technologies
- Capability Maturity Model Integration (CMMI)
- Integrated Systems Engineering, Test, & Supportability Disciplines
- Network Centric Operations
- Application of DoD Initiatives:
  - Performance-Based Business Environment
  - System Safety
  - Open Systems
  - Simulation-Based Acquisition
  - COTS Integration
- Systems Engineering Effectiveness
- Modeling & Simulation
- Integrated Risk Management
- Performance Based Logistics
- Improved Cycle Times for Design, Manufacture & Repair Process
- Improved Mission Readiness & Systems Availability
- Systems Engineering Training & Education

Brochure Contents
Hotel/Registration/Display/General Conference Information ....................... pg. 2-3
Conference Agenda at a glance ........ pg. 5
Tutorial Tracks .............................. pg. 7
Concurrent Session Listings .............. pg. 8-12
Conference Registration Form ........... pg. 13
Displayer Registration Form ............. pg. 14

For questions regarding the Conference, please contact: Britt Bommelje, Associate Director, at 703/247-2587 or bbommelje@ndia.org.
Hotel Information

Hyatt Regency Islandia
1441 Quivira Road
San Diego, California 92109
(619) 224-1234

Please call the hotel directly in order to make your reservation. A block of rooms have been reserved at the Hyatt Regency Islandia. In order to ensure the discounted NDIA rate, you must make your reservations early, and ask for the NDIA room block. Rooms will not be held after Friday, September 29, 2006, and may sell out before then. Rates are also subject to increase after this date. The government per diem rate is available only to active duty or civilian government employees. ID will be required upon check-in. Retired military ID's do not qualify.

Government Rate: $129.00*
Industry Rate: $169.00
*(or prevailing government per diem rate at the time of the Conference)

For more information about hotel and local attractions, please visit the hotel website at: http://islandia.hyatt.com/hyatt/hotels/index.jsp.

Registration Information

To register online for this Conference, please visit the following link: http://www.ndia.org/meetings/7870. Online registration will close on October 6, 2006. You must register onsite after this date. You can also download the registration form from the NDIA website or complete the form contained in this brochure. Fax the completed form to 703-522-1885 or mail to Event #7870, National Defense Industrial Association, 2111 Wilson Boulevard, Suite 400, Arlington, VA 22201-3061. Please do not fax or mail registration froms after October 6, 2006. You will need to register onsite after this date. Payment MUST be made at the time of registration. Registrations will not be taken over the phone. For additional questions, contact Britt Bommelje, Associate Director at 703-247-2587, or via email at bbommelje@ndia.org. Please see page 13 for the Conference Registration form.

CANCELLATIONS REMINDER: Cancellations received before August 30, 2006 will receive a full refund. Cancellations received from August 30, 2006 until October 6, 2006 will receive a refund minus a cancellation fee of $75.00. REFUNDS WILL NOT BE GIVEN FOR CANCELLATIONS RECEIVED AFTER OCTOBER 6, 2006. Substitutions are welcome in lieu of cancellations.
Displayer Information

NDIA invites you to display at the 9th Annual Systems Engineering Conference. An area will be available for the set-up of Corporate or Organization displays to demonstrate your company/organization's capability in Systems Engineering and Design expertise, Logistics Support, Systems Engineering and Design Tools, Modeling & Simulation Tools & Capability, expertise in Test & Evaluation, Reliability Analysis Tools & Capability, Process Improvement Capability & Tools including CMMI Appraisal Services & Tools, Engineering Services and related items. Displays will be table-top or “pop-up” style. Allocated display space will be 10 ft wide by 6 ft deep. Pipe and drape is not provided. Please see page 14 for the Display Registration Form.

For additional information on displays, please contact Britt Bommelje, at 703-247-2587 or at bbommelje@ndia.org.

General Conference Information

Identification Badges: During Conference registration and check-in, each participant will be issued an identification badge. Please be prepared to present a picture ID. Badges must be worn at all conference functions.

Conference Proceedings: Proceedings will be available on the web through the Defense Technical Information Center (DTIC), and will be available two to three weeks after the Conference. You will receive notification via e-mail that proceedings are posted and available on the web.

Promotional Partnership Opportunities: Increase your company or organization's exposure at this premier event by becoming a Promotional Partner. A Promotional Partnership will add your company name to the back cover of the on-site brochure as well as main platform recognition throughout the conference, signage at all events including the opening reception, a 350-word organization description in the on-site brochure, and a hotlink from the Conference webpage to your company website. For more information, please contact Sam Campagna at 703-247-2544 or via e-mail at scampagna@ndia.org.

Attire: Appropriate dress for this Conference is business casual for civilians and class B uniform for military.

ADA: NDIA supports the Americans with Disabilities Act of 1990. Attendees with special needs should call 703-522-1820 and refer to Event # 7870 prior to October 6, 2006.

National Defense Magazine: Advertise in National Defense and increase your company’s exposure at this Conference! National Defense will be distributed to the attendees of this Conference and all other NDIA Conferences. For more information, please contact Dino Pignotti at 703-247-2582 or dpignotti@ndia.org.

Inquiries: For questions regarding the Conference, please contact Britt Bommelje, Associate Director at 703-247-2587 or at bbommelje@ndia.org.

For more information on the Conference, or to register online, please visit the Conference website at www.ndia.org/meetings/7870.
### Conference Agenda -- At a Glance

#### Sun, October 22, 2006
- **5:00 pm - 7:00 pm**  
  Registration for Tutorials and General Conference  
  (Tutorials are an additional $200.00 registration fee)

#### Mon, October 23, 2006
- **7:00 am - 5:00 pm**  
  Registration
- **7:00 am - 8:00 am**  
  Continental Breakfast for Tutorial Attendees  
  (Tutorials are an additional $200.00 registration fee)
- **8:00 am - 12:00 pm**  
  Tutorial Tracks  
  (Please refer to page 7 for Tutorial Schedule)
- **12:00 pm - 1:00 pm**  
  Buffet Lunch for Tutorial Attendees
- **1:00 pm - 5:00 pm**  
  Tutorial Tracks Continued
- **5:00 pm - 6:00 pm**  
  Reception in Display Area (Open to All Participants)

#### Tue, October 24, 2006
- **7:30 am - 5:00 pm**  
  Registration
- **7:30 am - 8:30 am**  
  Continental Breakfast
- **8:30 am - 8:35 am**  
  Introductions  
  Mr. Sam Campagna, Director, Operations, NDIA
- **8:35 am - 8:45 am**  
  Opening Remarks  
  Mr. Bob Rassa, Director, Systems Supportability, Raytheon; Chair,  
  Systems Engineering Division, NDIA
- **8:45 am - 9:45 am**  
  Keynote Address  
  Hon. James Finley, Deputy Under Secretary of Defense, Acquisition &  
  Technology
- **9:45 am - 10:15 am**  
  Break
- **10:15 am - 12:00 pm**  
  Plenary Session: Executive Panel  
  **Moderator:**  
  Mr. Mark Schaeffer, Director, Office of Under Secretary of Defense for  
  Acquisition, Technology and Logistics, Director, Systems and Software  
  Engineering  
  **Panelists:**  
  Mr. Carl Siel, ASN/RDA Chief Engineer (Acting)  
  Mr. Terry Jaggers, SAF/AQ, Director, SAF/AQR (Science, Technology  
  & Engineering) (invited)  
  Mr. Doug Wiltsie, Assistant Deputy for ASM, HQDA, OASA (ALT) (invited)  
  Mr. Steven Kapurch, Chief Systems Engineer, NASA (invited)

Register now!! Visit the conference website at [http://www.ndia.org/meetings/7870](http://www.ndia.org/meetings/7870)
12:00 pm - 1:30 pm  Luncheon Awards Ceremony
Regency Annex

1:30 pm - 5:00 pm  Concurrent Sessions (Please refer to pages 8-12 for session schedule)

5:00 pm - 7:00 pm  Reception in Display Area

• Wed, October 25, 2006 • • Wed, October 25 • • Wed, October 25 • • Wed,

7:00 am - 5:15 pm  Registration

7:00 am - 8:00 am  Continental Breakfast

8:15 am - 12:00 pm  Concurrent Sessions (Please refer to pages 8-12 for session schedule)

12:00 pm - 1:30 pm  Luncheon Speaker
Regency Annex
Mr. Patrick Michael Kern, Deputy Assistant Secretary of Defense, NII/DoD CIO for Enterprise Wide Systems Engineering

2:30 pm - 5:30 pm  Concurrent Sessions (Please refer to pages 8-12 for session schedule)

• Thurs, October 26, 2006 • • Thurs, October 26 • • Thurs, October 26 • • Thurs, October 26 • • Thurs,

7:00 am - 3:00 pm  Registration

7:00 am - 8:00 am  Continental Breakfast

8:00 am - 12:00 pm  Concurrent Sessions (Please refer to pages 8-12 for session schedule)

12:00 pm - 1:00 pm  Luncheon

1:00 pm - 3:00 pm  Concurrent Sessions (Please refer to pages 8-12 for session schedule)

3:00 pm  Conference Adjourns

“The Department of Defense finds this event meets the minimum regulatory standards for attendance by DoD employees. This finding does not constitute a blanket approval or endorsement for attendance. Individual DoD component commands or organizations are responsible for approving attendance of its DoD employees based on mission requirements and DoD regulations.”

Register now!! Visit the conference website at http://www.ndia.org/meetings/7870 6
### Monday, October 23, 2006 - Tutorial Sessions

<table>
<thead>
<tr>
<th>Time</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7:00 am - 9:45 am</td>
<td><strong>Registration &amp; Continental Breakfast</strong></td>
</tr>
</tbody>
</table>
| 10:15 am - 12:00 pm | **Track 1 Tutorial**  
Session 1A1: Practical Six Sigma Tools for Systems Engineering  
Mr. Rick Hefner                                |
| 1:00 pm - 2:45 pm | **Track 1 Tutorial**  
Session 1B1: Practical Six Sigma Tools for Systems Engineers (Continued)  
Mr. Rick Hefner                                |
| 3:15 pm - 5:30 pm | **Track 1 Tutorial**  
Session 1C1: Performance (Performance and Reliability) Modeling  
Dr. Meng-Lai Yin                                |
| 5:30 pm - 6:30 pm | **Buffet Lunch**                                                            |
| 3:15 pm - 5:30 pm | **Track 2 Tutorial**  
Session 1A2: Requirements Engineering Workshop: A Practical Approach to Modeling and Managing Requirements  
Mr. William Deibler, II                          |
| 4:30 pm - 6:30 pm | **Track 2 Tutorial**  
Session 1B2: Requirements Engineering Workshop: A Practical Approach to Modeling and Managing Requirements (Continued)  
Mr. William J. Deibler, II                      |
| 5:30 pm - 6:30 pm | **Track 2 Tutorial**  
Session 1C2: Integrated Project Management (IPM) – The CMMI and Collaborative Product Development  
Mr. William Deibler, II                          |
| 3:15 pm - 5:30 pm | **Track 3 Tutorial**  
Session 1A3: How to Define Lean Systems Engineering Processes  
Mr. Tim Olson                                    |
| 4:30 pm - 6:30 pm | **Track 3 Tutorial**  
Session 1B3: How to Define Lean Systems Engineering Processes (Continued)  
Mr. Tim Olson                                    |
| 5:30 pm - 6:30 pm | **Track 3 Tutorial**  
Session 1C3: How to Measurably Improve Your Requirements  
Mr. Tim Olson                                    |
| 3:15 pm - 5:30 pm | **Track 4 Tutorial**  
Session 1A4: What You Need to Know About Chaos, Complexity, and Complex Adaptive Systems to do Systems Engineering Well in the 21st Century  
Ms. Sarah Sheard                                  |
| 4:30 pm - 6:30 pm | **Track 4 Tutorial**  
Session 1B4: What You Need to Know About Chaos, Complexity, and Complex Adaptive Systems to do Systems Engineering Well in the 21st Century (Continued)  
Ms. Sarah Sheard                                  |
| 5:30 pm - 6:30 pm | **Track 4 Tutorial**  
Session 1C4: Leading Effective Technical Reviews and Meetings  
Mr. David Walden, CSEP                            |
| 3:15 pm - 5:30 pm | **Track 5 Tutorial**  
Session 1A5: Systems Engineering Applications in Support of the Joint Capabilities Integration and Development System (JCIDS)  
Mr. Christopher Ryder                             |
| 4:30 pm - 6:30 pm | **Track 5 Tutorial**  
Session 1B5: Systems Engineering Applications in Support of the Joint Capabilities Integration and Development System (JCIDS) (Continued)  
Mr. Christopher Ryder                             |
| 5:30 pm - 6:30 pm | **Track 5 Tutorial**  
Session 1C5: Improving Systems Engineering Effectiveness Through Assessment and Process Improvement  
Mr. Ian Talbot                                    |
| 3:15 pm - 5:30 pm | **Track 6 Tutorial**  
Session 1A6: Integrated Specification Development Algorithm  
Mr. Jeffrey Grady                                  |
| 4:30 pm - 6:30 pm | **Track 6 Tutorial**  
Session 1B6: Integrated Specification Development Algorithm (Continued)  
Mr. Jeffrey Grady                                  |
| 5:30 pm - 6:30 pm | **Track 6 Tutorial**  
Session 1C6: Integrating Systems Engineering with Earned Value Management  
Mr. Paul Solomon                                  |
| 3:15 pm - 5:30 pm | **Track 7 Tutorial**  
Session 1A7: System Level Configuration Management  
Mr. Al Florence                                   |
| 4:30 pm - 6:30 pm | **Track 7 Tutorial**  
Session 1B7: System Level Configuration Management (Continued)  
Mr. Al Florence                                   |
| 5:30 pm - 6:30 pm | **Track 7 Tutorial**  
Session 1C7: Introduction to SysML & Object Oriented Systems Engineering Methodology (OOSEM)  
Dr. Abe Melich                                     |
| 3:15 pm - 5:30 pm | **Track 8 Tutorial**  
Session 1A8: QUASAR: A Method for Assessing the Quality of the Architecture of Large and Complex Software-Intensive Systems  
Mr. Donald Friesmith                             |
| 4:30 pm - 6:30 pm | **Track 8 Tutorial**  
Session 1B8: QUASAR: A Method for Assessing the Quality of the Architecture of Large and Complex Software-Intensive Systems (Continued)  
Mr. Donald G. Friesmith                           |
| 5:30 pm - 6:30 pm | **Track 8 Tutorial**  
Session 1C8: Acoustic Rapid COTS Insertion and Advanced Processing Build – Integrating Test & Evaluation Philosophies into Systems Engineering  
Mr. George Mars                                    |
| 3:15 pm - 5:30 pm | **Track 9 Tutorial**  
Session 1A9: Acoustic Rapid COTS Insertion and Advanced Processing Build – Integrating Test & Evaluation Philosophies into Systems Engineering (Continued)  
Mr. George Mars                                    |
| 4:30 pm - 6:30 pm | **Track 9 Tutorial**  
Session 1B9: Acoustic Rapid COTS Insertion and Advanced Processing Build – Integrating Test & Evaluation Philosophies into Systems Engineering (Continued) (Continued)  
Mr. George Mars                                    |
| 5:30 pm - 6:30 pm | **Track 9 Tutorial**  
Session 1C9: Acoustic Rapid COTS Insertion and Advanced Processing Build – Integrating Test & Evaluation Philosophies into Systems Engineering (Continued) (Continued) (Continued)  
Mr. George Mars                                    |

**Breaks:**
- 10:45 am - 11:15 am
- 1:00 pm - 1:30 pm
- 3:45 pm - 4:15 pm
<table>
<thead>
<tr>
<th>Track 1</th>
<th>Systems Engineering Effectiveness</th>
<th>Tuesday, October 24, 2006</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regency A</td>
<td>Mr. Al Brown</td>
<td>Defense SE Revitalization – A Work in Progress</td>
</tr>
<tr>
<td>Session 2C1</td>
<td>Mr. Robert Skalamera</td>
<td>Understanding the Technical Rationale and Intent of Requirements</td>
</tr>
<tr>
<td>Regency B</td>
<td>Mr. John Lobis</td>
<td>Assessing Technology Maturity Through Testing</td>
</tr>
<tr>
<td>Session 2C2</td>
<td>Mr. Chris DiPietto</td>
<td>A Closer Look at the Housing Bubble and Requirements Management</td>
</tr>
<tr>
<td>Regency C</td>
<td>Mr. Joseph E. Tribble</td>
<td>TRACK2 Test &amp; Evaluation in Systems Engineering</td>
</tr>
<tr>
<td>Mission A</td>
<td>Mr. David Han</td>
<td>Cost Efficient Risk Management through Integrated Life Cycle Management</td>
</tr>
<tr>
<td>Session 2C4</td>
<td>Mr. Edward Beck</td>
<td>TRACK3 Cost Management, Cost Engineering</td>
</tr>
<tr>
<td>Mission B</td>
<td>Mr. Jim Hollenbach</td>
<td>TRACK4 Systems Engineering – The Defense Acquisition System’s Environmental Management System</td>
</tr>
<tr>
<td>Session 2C6</td>
<td>Mr. Clifford Sammons</td>
<td>Implementing CMMI in an SE Organization</td>
</tr>
<tr>
<td>Island A</td>
<td>Dr. Paul Hefner</td>
<td>TRACK5 Best Practices &amp; Standardization</td>
</tr>
<tr>
<td>Session 2C7</td>
<td>Ms. Kristen Baldwin</td>
<td>Implementing a Standardized Lean and Paperless Change Management Process in a Legacy, Multi-Program Environment</td>
</tr>
<tr>
<td>Island B</td>
<td>Dr. Tom Christian</td>
<td>TRACK6 Software Engineering by Rick Tindall</td>
</tr>
<tr>
<td>Session 2C8</td>
<td>Dr. Mary Anne Herron</td>
<td>How Applying the Team Software Process (TSP) and Personal Software Process (PSP) can increase Software Supportability</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Track 2</th>
<th>Systems Engineering Effectiveness</th>
<th>Tuesday, October 24, 2006</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regency A</td>
<td>Mr. Jacob Horowitz</td>
<td>TRACK1 Systems Engineering Effectiveness</td>
</tr>
<tr>
<td>Session 2D1</td>
<td>Mr. Al Linsky</td>
<td>Systems Engineering Used in Venus Satellite Design</td>
</tr>
<tr>
<td>Regency B</td>
<td>Mr. William Lyders</td>
<td>TRACK2 Test &amp; Evaluation in Systems Engineering</td>
</tr>
<tr>
<td>Session 2D2</td>
<td>Parvaneh Safaei</td>
<td>System Engineers Provide a Key Contribution and Role in System Integration &amp; Test</td>
</tr>
<tr>
<td>Regency C</td>
<td>Mr. David Castellano</td>
<td>TRACK3 Program Management</td>
</tr>
<tr>
<td>Session 2D3</td>
<td>Mr. Sherman Forbes</td>
<td>TRACK4 Model-Based SE and Verification: Linking the “Vee”</td>
</tr>
<tr>
<td>Mission B</td>
<td>Mr. Patrick Young</td>
<td>TRACK5 Modeling &amp; Simulation</td>
</tr>
<tr>
<td>Session 2D4</td>
<td>Mr. Dennis Lowenstein</td>
<td>TRACK6 Net Centric Operations</td>
</tr>
<tr>
<td>Mission C</td>
<td>Dr. Robert Mills</td>
<td>TRACK7 Best Practices &amp; Standardization</td>
</tr>
<tr>
<td>Session 2D6</td>
<td>Ms. Satya Mohanty</td>
<td>TRACK7 Software Engineering – The Defense Acquisition System’s Environmental Management System</td>
</tr>
<tr>
<td>Island A</td>
<td>Dr. Mary Anne Herron</td>
<td>TRACK8 Software Engineering by Rick Tindall</td>
</tr>
<tr>
<td>Session 2D8</td>
<td>Mr. Jon Peterson</td>
<td>How Applying the Team Software Process (TSP) and Personal Software Process (PSP) can increase Software Supportability</td>
</tr>
</tbody>
</table>

**Break in Display Area**

**5:30 pm - 7:00 pm Reception in Display Area**
Wednesday, October 25, 2006

7:00 am Registration & Continental Breakfast

8:00 am - 9:45 am

TRACK 1 Systems Engineering Effectiveness
Mr. Al Brown Session 3A1
Mr. James Miller
Mr. Jeff Loren

Mr. John Lohse Session 3A2
Mr. Raymond Beach Dr. Robert Franceschini

Mr. Bob Lyons Session 3A3
HON Gary Payton Ms. Rebecca Cowen-Hirsch

Mr. Howard Savage Session 3A4 Dr. James Dill

Mr. Jim Hollenbach Session 3A5 Mr. David Henry Mr. Kevin Tang

Mr. Jack Zavin Session 3A6 Dr. Daniel Leshem Mr. Jeffrey Levin

Mr. Paul Croll Session 3A7 Ms. Cynthia Hauer Mr. Tim Olson

Mr. Mark Pellegrini Mr. Michael Goy

10:15 am - 12:00 pm

TRACK 1 Systems Engineering Effectiveness
Mr. Al Brown Session 3B1 Mr. Jacob Herskovitz Mr. Dennis Cooper

Mr. John Lohse Session 3B2 Mr. Scott Rideout Mr. Dean Carico

Mr. Bob Lyons Session 3B3 Dr. Jay Mandelbaum Mr. Ed Casey

Mr. Chuck Silva Session 3B4

Mr. Jim Hollenbach Session 3B5 Mr. Frank Salvatore Mr. Scott Holben

Mr. Jack Zavin Session 3B6 Mr. David Emery Mrs. Colleen Cannon

Mr. Paul Croll Session 3B7 Dr. David Walden, CSEP Ms. Jeanne Balsam

Mr. Mark Pellegrini Mr. Michael Goy

12:00 pm Lunch in the Regency Annex
<table>
<thead>
<tr>
<th>Time</th>
<th>Session</th>
<th>Topic</th>
<th>Presenter(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3:30 pm</td>
<td>Session 1</td>
<td>On The Derivation of a Program Total Status Metric with Statistical Correlation</td>
<td>Mr. James Lackey</td>
</tr>
<tr>
<td>3:30 pm</td>
<td>Session 2</td>
<td>Concept for an Enterprise Wide System Engineering Collaborative Environment</td>
<td>Mr. Al Brown</td>
</tr>
<tr>
<td>3:30 pm</td>
<td>Session 3</td>
<td>Effectiveness Mr. Al. Brown Session 2C1</td>
<td>Mr. Al Brown</td>
</tr>
<tr>
<td>3:30 pm</td>
<td>Session 4</td>
<td>Effectiveness Mr. Al. Brown Session 2C2</td>
<td>Mr. Al Brown</td>
</tr>
<tr>
<td>3:30 pm</td>
<td>Session 5</td>
<td>Effectiveness Mr. Al. Brown Session 2C3</td>
<td>Mr. Al Brown</td>
</tr>
<tr>
<td>3:30 pm</td>
<td>Session 6</td>
<td>Effectiveness Mr. Al. Brown Session 2C4</td>
<td>Mr. Al Brown</td>
</tr>
<tr>
<td>3:30 pm</td>
<td>Session 7</td>
<td>Effectiveness Mr. Al. Brown Session 2C5</td>
<td>Mr. Al Brown</td>
</tr>
<tr>
<td>3:30 pm</td>
<td>Session 8</td>
<td>Effectiveness Mr. Al. Brown Session 2C6</td>
<td>Mr. Al Brown</td>
</tr>
</tbody>
</table>

**Break in Display Area**
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Regency A</td>
<td>Mr. Al Brown</td>
<td>Dr. William Thomas</td>
<td>Mr. Michael Kutch</td>
<td>Mr. Doug Atkinson</td>
</tr>
<tr>
<td>Regency B</td>
<td>Mr. Al Brown</td>
<td>Mr. Paul Croll</td>
<td>Ms. Michele Moss</td>
<td>Ms. Elizabeth Miller</td>
</tr>
<tr>
<td>TRACK 3</td>
<td>Program Management</td>
<td>Effective QA on Small Projects</td>
<td>Systems Management: Can It Help Forecast Future Program Performance?</td>
<td></td>
</tr>
<tr>
<td>Regency C</td>
<td>Mr. Bob Lyons</td>
<td>Mr. Jean Swank</td>
<td>Mr. Adrio DeCicco</td>
<td></td>
</tr>
<tr>
<td>TRACK 4</td>
<td>Education &amp; Training</td>
<td>Continuous Learning Module – M&amp;S for Systems Engineering</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mission A</td>
<td>Mr. Michael Truelove</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TRACK 5</td>
<td>Modeling &amp; Simulation</td>
<td>A Supply Chain Simulation for Program Managers in Manufacturing</td>
<td>Modeling &amp; Simulation in Systems Engineering - “Standardizing the Next Generation of Military Vehicle Cooling Systems”</td>
<td></td>
</tr>
<tr>
<td>Mission B</td>
<td>Mr. Jim Hollenbach</td>
<td>Mr. Swen Leong</td>
<td>Mr. Neil Slyva</td>
<td></td>
</tr>
<tr>
<td>TRACK 6</td>
<td>Net Centric Operations</td>
<td>An Industry Situational Assessment of the Current Maturity of Data Standards, Processes and Infrastructure to Implement a Net-Centric Information Systems Common Data Environment</td>
<td>Determining System Interoperability Using an Integration Readiness Level</td>
<td></td>
</tr>
<tr>
<td>Mission C</td>
<td>Mr. Jack Zawin</td>
<td>Mr. Dick Engwall</td>
<td>Dr. Brian Sauser</td>
<td></td>
</tr>
<tr>
<td>TRACK 7</td>
<td>Best Practices &amp; Standardization</td>
<td>Extending the Team Software Process (TSP) for Systems Engineering</td>
<td>Implementations of the STEP Systems Engineering Standard (AP233)</td>
<td></td>
</tr>
<tr>
<td>Island A</td>
<td>Mr. Paul Croll</td>
<td>Mr. Dennis Linck</td>
<td>Mr. James U'Ren</td>
<td></td>
</tr>
<tr>
<td>TRACK 8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Island B</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Session 4C8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Conference Adjourns
### Registration Fees

<table>
<thead>
<tr>
<th></th>
<th>Early</th>
<th>Regular</th>
<th>Late</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>before 08/30/06</td>
<td>8/30/06-10/06/06</td>
<td>10/06/06</td>
</tr>
<tr>
<td>Government/Academia</td>
<td>$660</td>
<td>$725</td>
<td>$800</td>
</tr>
<tr>
<td>Industry/NDIA Member</td>
<td>$775</td>
<td>$855</td>
<td>$940</td>
</tr>
<tr>
<td>Industry/Non-Member</td>
<td>$825</td>
<td>$910</td>
<td>$1000</td>
</tr>
</tbody>
</table>

**Tutorial Registration** - MONDAY ONLY □ $200

*Tutorial registration is for the Monday Tutorial Session ONLY, and it is in addition to the Conference Registration Fee*

Cancellations received before August 12, 2006 will receive a full refund. Cancellations received between August 12, 2006 and October 6, 2006 will receive a refund minus a cancellation fee of $75. No refunds for cancellations received on or after October 6, 2006.

Substitutions welcome in lieu of cancellation.

### Payment Options

- Check (payable to NDIA)
- Cash
- Government PO/Training Form #
- VISA
- MasterCard
- American Express
- Diners Club

If paying by credit card, you may return by fax to (703) 522-1885.

### Questions?

Contact Britt Bommelje, Associate Director
(703) 247-2587 email: bbommelje@ndia.org

Mail to:
NDIA, Event #7870
2111 Wilson Boulevard, Suite 400
Arlington, VA 22201

Fax to:
(703)522-1885
Name ____________________________________________
Title ____________________________________________
Company Name ______________________________________
Division/Dept. ______________________________________
Address ____________________________________________
City/State/Zip ______________________________________
Phone ___________________________ Fax _________________
E-mail ____________________________________________

Display Requirements:
All displays must be of the simple table-top/pop-up style standards. Space per display shall not exceed 10 ft. wide by 6 ft. deep.
Minimal hardware to be utilized (computer systems for demonstrations are OK). No formal decorating company is involved.
Companies must bring their own displays and plan to do their own set-up. Standard 2.5 x 6 ft. draped folding tables and chair
will be provided for each display space. No other props or setups (pipe & drape, plants, etc.) will be utilized.

Display Hours:
Displays are to be set up by 4:00 PM October 23, 2006 and should remain in place until after the mid-morning break on October
26, 2006. Displays must be removed by 4:00 PM October 26, 2006.

Cost: Displays (includes one free registration and electrical hook-up for display): $ 1,000

Display Rules & Regulations:
1. If NDIA should be prevented from holding the conference for any reason beyond NDIA's control (such as, but not limited to,
damage to the building, riots, strikes, acts of government, or acts of God) or if a displayer cannot occupy the assigned display
space due to reasons beyond NDIA’s control, then NDIA has the right to cancel the conference or any part thereof, with no fur-
ther liability to the displayer other than a refund of display space fee, less a proportionate share of the conference cost incurred.
2. Neither the management of the host facility nor NDIA shall be liable for the damages, loss or destruction to the displays by
reason of fire, theft, accident or other destructive causes. Displayers shall lease space at his sole risk. Neither the management
of the host facility, NDIA nor any of their agents, servants or employees will be accountable or liable for accidents to displayers,
their agents or employees.
3. The displayer shall be liable to the host facility and/or NDIA for any damage to the building and/or the furniture and fixtures
contained therein which shall occur through acts or omissions of the displayer.
4. Displayers assumes the entire responsibility and hereby agrees to protect, indemnify and hold harmless NDIA, the
host facility, their officers, employees, and agents against all claims, losses and damages to persons and property, governmental
charges or fines, and attorney’s fees arising out of or caused by displayers installation, removal, maintenance, occupancy or use
of the display premises or any part thereof, including any outside display areas.
5. Displayers acknowledges that NDIA does not maintain and is not responsible for obtaining insurance covering displayers’s
property. Displayers are advised to obtain business interruption and property damage and loss insurance to cover such occur-
rences.

Send this form with payment for display to:
Britt Bommelje, Associate Director, National Defense Industrial Association, 2111 Wilson Boulevard, Suite 400, Arlington, VA
22201-3061, Phone: (703)247-2587, Fax: (703)522-1885, E-mail: bbommelje@ndia.org.

Deadline for sign-up is Friday, October 13, 2006. Make checks payable to NDIA - Event # 7870.

☐ Check (payable to NDIA - Event #7870)

☐ Visa   ☐ Diner’s Club   ☐ Mastercard   ☐ Amex

Credit Card # ___________________________ Exp. Date ___________________________
Authorized Signature ________________________________________________________
Looking for something to do in San Diego?

Ride the Giant Dipper Roller Coaster at Mission Beach

Explore the San Diego Zoo

Experience a San Diego Charger Football game - Sunday, October 29 1:05 pm vs. St. Louis Rams

Tour the USS Midway in San Diego’s Port

For more information on what to do in San Diego please visit www.sandiego.org
QUality Assessment of System ARchitectures (QUASAR)

Donald Firesmith
Acquisition Support Program (ASP)

Sponsored by the U.S. Department of Defense
© 2006 by Carnegie Mellon University
Topics

What is System Architecture?
Why is System Architecture Critical?
Why Assess the System Architecture?

QUASAR System Architecture Assessment Method:
• Philosophy
• Quality Cases
• QUASAR Process
What is a System Architecture?

Traditional Definition:
the fundamental structure of a system in terms of its major components, their relationships to each other and the system’s environment, and the principles governing the creation and evolution of the structure

More General Definition:
the most important, pervasive, top-level, strategic inventions, decisions, and their associated rationales about the system including its overall structure (i.e., essential architectural elements, their relationships, and their associated blackbox characteristics and behavior)
Architecture vs. Design

Architecture Decisions:
• Pervasive across System Components
• Strategic Decisions and Inventions
• Higher-Levels of System
• Huge Impact on Quality, Cost, and Schedule
• Drives Design, Highest-Level Integration, and Integration Testing
• Driven by Requirements and Higher-Level Architecture
• Mirrors Top-Level Organization of Development Team

Design Decisions:
• Local within Individual System Components
• Tactical Decisions and Inventions
• Lower-Levels of System
• Smaller Impact on Quality, Cost, and Schedule
• Drives Implementation, Lowest-Level Integration, and Unit Test
• Driven by Requirements, Architecture, and Higher-Level Design
Why is Architecture Critical?

Architecture Defines:
• Key System Components
• How Key Components Interact

Architecture Affects:
• Design Decisions
• Implementation Decisions
• Integration Decisions
• Testing Decisions

Architectural Decisions Drive:
• Ultimate System Quality
• Development Costs
• Development Schedule
• Sustainment Costs
• Maintenance and Upgradeability
Why is Architectures Critical?

The quality of the architecture drives the quality of the system:

- Availability
- Interoperability
- Modifiability
- Performance
- Reliability
- Robustness (Error, Failure, and Fault Tolerance)
- Safety
- Security
- Scalability
- Stability
- Testability
- …
Why Assess the Architecture?

Determine System Architecture:
- Quality
- Maturity and Completeness
- Integrity and Consistency
- Usability

Determine Compliance:
- Contract Compliance
- Requirements Compliance

Early Identification of System Architecture Defects:
- Fix Defects Early
- Decrease Costs
- Decrease Schedule
Why Assess the Architecture?

Manage Risks:
• System Architecture Risks
• System Risks

Provide Acquirer Oversight into System Architecture

Develop Consensus:
• Among Developers
• Between Acquirer and Developer Organization

Ensure Specification of Quality Requirements

Help Architects Succeed

Help Program Succeed
How to Assess the Architecture?

Assessment Philosophy
Quality Cases as Foundation

QUASAR Process:
• Phases
  - System Architecture Assessment Initiation
  - Subsystem Requirements Review
  - Subsystem Architecture Assessment
  - System Architecture Assessment Summary
Assessment Philosophy

Quality Requirements Drive the System Architecture.
Architects should Make Case to Assessors:
- Architects Know Quality Requirement Drivers
- Architects Know What they Did and Why
- Architects Know Where Documented

Safety Cases can Generalize into **Quality Cases**
(a.k.a., assurance cases) consisting of:
- **Claims**: Architecture Supports Quality Requirements
- **Arguments**: Architects’ Architectural Decisions and Rationales
- **Evidence**: Architects’ Documentation and Witnessed Demonstrations
Assessment Philosophy

Arguments must be Clear and Compelling.
Evidence must Be Credible.
Architects’ Responsibilities:
• Prepare Quality Cases
• Provide Early Presentation Materials to Assessors
• Present Quality Cases (Make Case to Assessors)
• Answer Assessors’ Questions
Assessor Responsibilities:
• Prepare for Assessments
• Probe Quality Cases
• Determine and Report Assessment Results
Quality Cases – Quality Model

Quality of a system (and system architecture) is defined in terms of a quality model:

1. Quality Model defines the meaning of quality for the
2. Quality Factor defines a type of the quality of the
3. Quality Subfactor defines a part of a type of the quality of the
4. Quality Measure (Measurement Scale) is measured using a

System
Quality Cases – Quality Factors

Quality Model

- Quality Factor
  - Development-Oriented Quality Factor
  - Usage-Oriented Quality Factor

Quality Subfactor

- Safety
- Security
- Defensibility
- Robustness
- Correctness
- Soundness
- Operational Availability
- Predictability
- Reliability
- Stability
- Efficiency
- Interoperability
- Performance
- Utility
- Dependability
- Capacity
- Configurability

Quality Measure (Measurement Scale)

is measured using a

Performance

Utility
Quality Cases – Quality Subfactors

- Harm
- Accident & Safety Incident
- Nonmalicious Agent
- Internal Vulnerability
- Hazard
- Safety Risk

- Safety Problem Type
- Safety Solution Type
- Prevention
- Detection
- Reaction
- Adaptation

Safety Subfactor
Quality Cases - Components

1. Claims
   Their architecture adequately supports its derived and allocated quality goals and requirements

2. Clear and compelling Arguments
   • Architecture decisions
   • Associated rationales

3. Supporting Evidence
   • Official program documentation
   • Witnessed demonstrations

Simplified version of safety case from safety community
Quality Cases - Relationships

- Quality Subfactor
  - defines a part of a type of quality of a
  - is specific to a
    - Quality Factor
      - defines a type of quality of a
        - Quality Case
          - makes the case for the quality of a
            - System
              - Subsystem

- Claim
  - justifies belief in
- Argument
  - supports
- Evidence
Architecture Quality Cases

Architecture Quality Case

Quality Case

Claim

Architectural Claim

Argument

Architectural Argument

Evidence

Architectural Evidence

System

Subsystem

Architecture

makes the case for the quality of an

has an

makes the case for the quality of a

justifies belief in

supports

justifies belief in

supports

justifies belief in
Interoperability (Quality) Case
Quality Case Diagram

Quality Cases contain a large amount of Information. Claims, Arguments, and a large amount of Evidence are typically text. It is easy to get lost in a large, complex, textual quality case. A quality Case Diagram is a layered UML class diagram that labels and summarizes the parts of a single quality case:

- **Claims:**
  - Quality Goals
  - Quality Requirements
- **Arguments:**
  - Architectural Decisions
  - Rationale
- **Evidence:**
  - Documentation
  - Witnessed Demonstrations
Partial Performance Quality Case Diagram

Goal:
Architecture Supports Performance
<<claim>>

Goal:
Architecture Limits Jitter
<<claim>>

Requirement:
Architecture Limits Jitter
<<claim>>

Architecture Decision:
Real-Time Operating System
<<argument>>

Goal:
Architecture Supports Schedulability
<<claim>>

Requirement:
Architecture Supports Schedulability
<<claim>>

Architecture Decision:
Deterministic Scheduling
<<argument>>

Goal:
Architecture Limits Response Time
<<claim>>

Requirement:
Architecture Limits Response Time
<<claim>>

Goal:
Architecture Limits Latency
<<claim>>

Requirement:
Architecture Limits Latency
<<claim>>

Goal:
Architecture Supports Throughput
<<claim>>

Requirement:
Architecture Supports Throughput
<<claim>>

Architecture Decision:
Layered Architecture
<<argument>>

Architecture Decision:
Redundant Hardware
<<argument>>

Architecture Decision:
Load Balancing
<<argument>>

justifies belief in
QUASAR Assessment Process

Four Phases:
1. System Architecture Assessment Initiation (SAAI)
   For each Subsystem to be assessed:
2. Subsystem Requirements Review (SRR)
3. Subsystem Architecture Assessment (SAA)
4. System Architecture Assessment Summary (SAAS)

Each Phase consists of 3 Tasks:
1. Preparation
2. Meeting
3. Follow-Through
QUASAR Phases

System Architecture Assessment Initiation

repeat for each subsystem being assessed

Subsystem Requirements Review → Subsystem Architecture Assessment

done

yes

no

System Architecture Assessment Summary
QUASAR Phases and Tasks

<table>
<thead>
<tr>
<th>System Architecture Assessment Initiation Phase</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prep.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Subsystem Architecture Assessment Phase</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prep.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Subsystem Architecture Assessment Phase Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prep.</td>
</tr>
</tbody>
</table>
Quasar Teams

- **Architecturally Significant (Quality) Requirements**
  - Produce the Architecture
  - Drive the Architecture
  - Assess the quality of the Architecture

- **System Architecture**
  - Produce the System Architecture
  - Lead the Subsystem Architecture Teams

- **Subsystem Architecture**
  - Produce the Subsystem Architecture

- **Top-Level Architecture Team**
  - Make their decisions

- **Assessment Team(s)**
  - Evaluate the architects’ work

- **Quality Cases**
  - Make their decisions

© 2006 by Carnegie Mellon University

Version 0.2
System Architecture Assessment Initiation (SAAI) Phase

- System Architecture Assessment Initiation
- Repeat for each subsystem being assessed
- Subsystem Requirements Review
- Subsystem Architecture Assessment
- System Architecture Assessment Summary

Decision:
- No: Repeat
- Yes: Go to System Architecture Assessment Summary
SAAI Topics

SAAI Phase Objectives
SAAI Phase Principles
SAAI Phase Context
SAAI Phase Overview
  • SAAI Preparation Task
  • SAAI Meeting Task
  • SAAI Follow-Through Task
SAAI Roles and Responsibilities
Discussion
SAAI Phase Objectives

Prepare the teams

Develop Consensus:
  • Scope the Assessments
  • Schedule the Assessments
  • Tailor the Assessment Process and Training Materials
  • Capture Lessons Learned

Produce and Publish Meeting Outbrief and Minutes

Manage Action Items
SAAI Phase Principles

Need to Develop Consensus between Assessors and Assesses

Need to Tailor Process to meet specific Needs of the Overall Assessment

Scope of Assessment should match Project Needs and Resources

Subsystem Assessments must be scheduled to ensure required Resources
SAAI Preparation Task

Steps:

1. Staff Assessment Team
2. Train Assessment Team
3. Assessment Team Identifies the Top-Level Development Team (Top-Level Requirements Engineers & Architects)
4. Assessment Team Trains Top-Level Development Team
5. Teams Collaborate to Organize Meeting (attendees, time, location, agenda)
SAAI Meeting Task

Steps:

1. Teams Collaborate to Determine Assessment Scope:
   - Architecturally Significant Requirements
   - Subsystems
   - Assessment Resources

2. Teams Collaborate to Develop Initial Assessment Schedule

3. Teams Collaborate to Tailor Assessment Process

4. Assessment Team Manages Action Items
SAAI Follow-Through Task

Steps:
1. Assessment Team develops and presents Meeting Outbrief
2. Assessment Team develops, reviews, and distributes Meeting Minutes
3. Assessment Team tailors and distributes:
   - Assessment Procedure
   - Assessment Training Material
4. Assessment Team distributes Assessment Schedule
5. Teams obtain Needed Resources
6. Assessment Team captures Lessons Learned
7. Assessment Team Manages Action Items
SAAI Roles and Responsibilities

The system architecture assessment initiation phase is performed by the following teams:

- Assessment Team
- Top-Level Development Team:
  - Top-Level Requirements Team with input from Subsystem Requirements Teams
  - Top-Level Architecture Team with input from Subsystem Requirements Teams
Assessment Team Membership

Assessment Team Leader
Assessors
Meeting Facilitator
Subsystem Liaison
Subject Matter Experts
Scribe
Assessment Team Responsibilities

- Provide Architecture Assessment Training Materials
- Provide Architecture Assessment Procedure
- Collaborate to Tailor Architecture Assessment Procedure
- Collaborate to provide Initial Kick-Off Meeting Agenda
- Take Initial Kick-Off Meeting Notes
- Collaborate to Develop Assessment Schedule
- Produce Architecture Assessment Action Item List
- Produce Outbrief and Meeting Minutes
Development Team Membership

Top-Level Requirements Team:
• Lead Requirements Engineer
• System Requirements Engineers
• Subsystem Requirements Engineers

Top-Level Architecture Team:
• Lead System Architect
• System Architects
• Subsystem Architects
Development Team Responsibilities

Read Architecture Assessment Training Materials
Read Architecture Assessment Procedure
Collaborate to Tailor Architecture Assessment Procedure
Collaborate to provide Initial Kick-Off Meeting Agenda
Take Initial Kick-Off Meeting Notes
Collaborate to Develop Assessment Schedule
SAAI Discussion

What is the main objectives of the system architecture assessment initiation phase?
What are the three tasks comprising the SAAI phase?
What teams are involved?
What are the memberships and responsibilities of these teams?
Subsystem Requirements Review (SRR) Phase

1. System Architecture Assessment Initiation
2. Subsystem Requirements Review
3. Subsystem Architecture Assessment
   - repeat for each subsystem being assessed
4. System Architecture Assessment Summary
   - done
   - yes
   - no
SRR Topics

SRR Questions for the Attendees
SRR Phase Objectives
SRR Phase Principles
SRR Phase Challenges
SRR Phase Context
SRR Phase Overview
  • SRR Preparation Task
  • SRR Meeting Task
  • SRR Follow-Through Task
SRR Roles and Responsibilities
Discussion
SRR Questions for the Attendees

As a requirements engineer, what are your biggest problems with respect to engineering (e.g., identifying, deriving, analyzing, specifying, and managing) system and subsystem requirements that significantly impact the architecture?

As a system architect, what are your biggest problems with respect to the system requirements that significantly impact the architecture?

As a subsystem architect, what are your biggest problems with respect to the derived architecturally significant requirements that have been allocated to your subsystem?
Key Questions for the Attendees

How can you know the architecture is ‘good enough’ if the requirements do not specify exactly how good it has to be?

• How else can the architects:
  - Make engineering trade-offs among the different quality factors?
  - Know when the architecture is done?
• How can the architecture assessors assess the quality of the architecture without having requirements against which to make the assessment?
• How can testers determine success or failure without measurable test completion criteria?
• How can managers know the quality and status of the architecture without measurable indicators?
SRR Phase Objectives

Ensure that the:

1. Architecturally significant requirements are properly engineered in time to support the engineering of the subsystem architecture.

2. Subsystem architects know how to prepare for and support the coming subsystem architecture quality assessment.
Architecturally Significant Requirements

Architecturally Significant Requirement
Any requirement that has a significant impact on the system architecture

Quality requirements are typically the most important architecturally significant requirements.

Definition
Any requirement that specifies a minimum level of quality
Quality Requirements

Format
The system shall do X with threshold Y under condition(s) Z.

Bad Example(s)
The system shall be highly reliable, robust, safe, secure, stable, etc.

Good Example (Stability)
The system shall ensure that the mean time between the failure of non-critical functionality* causing the failure of critical functionality* is least 5,000 hours of continuous operation under normal operating conditions*.

* Must be properly defined in the project glossary.
SRR Phase Principles

Not all requirements are architecturally significant. Quality requirements should be major drivers of the system architecture. Quality requirements should specify a minimum required amount of some type of quality.

Quality requirements should be:

- Unambiguous
- Feasible
- Complete
- Consistent
- Mandatory
- Verifiable
- Validatable
SRR Phase Principles

Quality requirements should be organized according to a quality model that defines quality factors (a.k.a., attributes, “ilities”) and their quality subfactors:

- Availability
- Interoperability
- Performance
  - Jitter, Response Time, Schedulability, and Throughput
- Portability
- Reliability
- Safety
- Security
- Usability
**SRR Phase Principles**

Different quality factors are important for different subsystems.
- Performance is paramount for some subsystems.
- Security is more important for other subsystems.

Engineering architecturally significant requirements is the responsibility of the requirements team, not the architecture team and not the assessment team.
- Architects and assessors are not qualified to engineer quality requirements.
- Many stakeholders need quality requirements.
- Architecture assessment time is too late to engineer quality requirements.
SRR Phase Challenges

Architects are rarely given/allocated a *complete* set of architecturally significant requirements. These architecturally significant requirements rarely include quality requirements for *all* of the relevant quality factors and subfactors.
SRR Phase Challenges

The quality of the derived and allocated architecturally significant requirements are typically poor:
  • Requirements are often ambiguous.
    - “The system shall be safe and secure.”
  • Requirements rarely specify thresholds on relevant quality measurement scales.
    - “The system shall have adequate availability.”
  • Requirements are often mutually inconsistent.
    - Security vs. usability, performance vs. reliability.
  • Many requirements are infeasible (or at least impractical) if taken literally.
    - “The system shall have 99.99999 reliability.”
SRR Phase Challenges

Requirements are often unstable.

Specialty engineering requirements (e.g., reliability, safety, security) are often documented separately from the functional requirements.

The architecturally significant requirements are often *improperly* prioritized for implementation.

The subsystem architects often do not understand how to prepare for an architecture assessment.

- Too busy
- Not trained
- No standards exist
- Bias against assessments/audits
The subsystem architects do not understand how to give the assessment team the information they need to assess the architecture:

- How good must the architecture be to sufficiently support its derived and allocated quality requirements (i.e., to ‘pass’ the assessment)?
- What architectural decisions did the architects make to support the quality goals and requirements?
- What were the rationales for these decisions?
- What is the official documentation of actual architectural decisions?
  - Not plans and procedures
  - Official program documentation
  - Not hastily produced PowerPoint slides
SRR Phase Context

System Architecture Assessment Initiation Phase

<table>
<thead>
<tr>
<th>Prep.</th>
<th>Initial Meeting</th>
<th>Follow Through</th>
</tr>
</thead>
</table>

Subsystem Architecture Assessment Phase

<table>
<thead>
<tr>
<th>Subsystem 1</th>
<th>Subsystem Requirements Review Phase</th>
<th>Subsystem Architecture Assessment Phase</th>
</tr>
</thead>
</table>

Subsystem 2 Architecture Assessment

<table>
<thead>
<tr>
<th>Subsystem 2</th>
<th>Subsystem Requirements Review Phase</th>
<th>Subsystem Architecture Assessment Phase</th>
</tr>
</thead>
</table>

Subsystem N Architecture Assessment

<table>
<thead>
<tr>
<th>Subsystem N</th>
<th>Subsystem Requirements Review Phase</th>
<th>Subsystem Architecture Assessment Phase</th>
</tr>
</thead>
</table>

Time (not to scale)
SRR Phase Overview
SRR Preparation Task

Steps:

1. Subsystem Requirements Team provides access to the architecturally significant subsystem requirements as well as a summary of these requirements

2. Subsystem Architecture Team provides sample of planned Quality Cases

3. Subsystem Assessment Team reviews this information prior to the meeting
SRR Meeting Task

Steps:

1. Subsystem Requirements Team presents Summary of the architecturally significant subsystem requirements (organized by quality factor and quality subfactors)
2. Subsystem Assessment Team recommends Improvements
3. Subsystem Architecture Team presents sample of planned Quality Cases
4. Subsystem Assessment Team recommends Improvements
5. Assessment Team Manages Action Items
SRR Follow-Through Task

Steps:

1. Subsystem Assessment Team presents Outbrief
2. Subsystem Assessment Team develops and publishes Meeting Minutes containing recommendations for improving:
   - Architecturally significant subsystem requirements
   - Quality Cases
3. Assessment Team tailors and distributes updated Assessment Procedure and Assessment Training Material (for future requirements reviews)
4. Assessment Team captures Lessons Learned
5. Assessment Team Manages Action Items
SRR Roles and Responsibilities

The subsystem requirements review phase is performed by the following three teams:

• Subsystem Requirements Team
• Subsystem Architecture Team
• Subsystem Assessment Team
Subsystem Requirement Team

Responsibilities:

• Work with specialty engineering teams to engineer the architecturally significant subsystem requirements
• Provide these requirements to the subsystem architecture team in time to drive the subsystem architecture
• Provide the subsystem assessment team with access to these requirements sufficiently prior to the meeting
• Summarize these requirements at the requirements review meeting
• Answer questions from the assessment team (and architecture team)
Subsystem Architecture Team

Responsibilities:

• Develop a proposed representative sample of the architectural information to be presented during the coming subsystem architecture assessment meeting:
  - Architectural decisions and rationale
  - Supporting evidence
• Present this information to the subsystem assessment team
• Ask questions (if necessary) of the:
  - Subsystem requirements team (regarding architecturally significant requirements)
  - Subsystem assessment team (regarding the assessment process and adequacy of proposed sample architectural decisions, rationale, and evidence)
Subsystem Assessment Team

Responsibilities:
- Review supplied information prior to the requirements review meeting
- Ensure that the architecturally significant requirements are adequately engineered to support the subsystem architecture assessment.
- Ensure that the proposed architectural information to will be adequate to support the coming subsystem architecture assessment meeting
- Answer questions from and provide advice to the:
  - Requirements team regarding the architecturally significant requirements
  - Architecture team regarding what will be expected of them during the coming subsystem architecture assessment meeting
Subsystem Assessment Team

Responsibilities:

- Must include members having expertise in:
  - Requirements engineering and quality requirements
  - The system architecture quality assessment method
    (with all members having been trained in the method)
- Should include members having experience in the subsystem application domain(s) such as avionics, sensors, or weapons
SRR Discussion Questions

What is the two main objectives of the subsystem requirements review?
How often should subsystem requirements reviews be performed?
When should subsystem requirements reviews be performed?
What are the three tasks comprising the subsystem requirements review?
What are the objectives of these three tasks?
What teams are involved?
What are the responsibilities of these teams?
Subsystem Architecture Assessment (SAA)

- System Architecture Assessment Initiation
  - Subsystem Requirements Review
  - Subsystem Architecture Assessment
    - repeat for each subsystem being assessed
      - done
      - no
        - System Architecture Assessment Summary
          - yes
SAA Topics

SAA Questions for the Attendees
SAA Phase Objectives
SAA Phase Principles
SAA Phase Challenges
SAA Phase Context
SAA Phase Overview:
  • SAA Preparation Task
  • SAA Meeting Task
  • SAA Follow-Through Task
SAA Roles and Responsibilities
Discussion
SAA Questions for the Attendees

As a subsystem architect, what are your biggest problems with respect to:

• Engineering the subsystem architecture?
• Ensuring that the subsystem architecture adequately meets its architecturally significant requirements?
• Internally reviewing/evaluating the quality of the subsystem architecture?
• Supporting independent assessments of the quality of your subsystem architecture?

As an independent assessor (e.g., PO of prime contractor, prime contractor of subcontractor), what are your biggest problems with respect to independently assessing the quality of an acquired subsystem’s architecture?
SAA Questions for the Attendees

Is the quality of your architectures being independently assessed?
How are your architectures being assessed?
Who is assessing your architectures?
What do you see as the biggest problems with respect to how your architectures are being assessed?
  • Are your assessors using an effective and efficient process for assessing your architectures?
  • Do you know what is expected of you during the system architecture assessments?
  • Do you develop adequate documentation as a natural part of the architecture process?
  • Is the architecture documentation you develop adequate to support assessments?
SAA Objectives

Assess Quality of Subsystem Architecture in terms of:

• Architectures support for its derived and allocated architecturally significant requirements

• Architectural Quality Cases
SAA Principles

Quality architecture assessments should be organized according to a quality model that defines quality factors (a.k.a., attributes, “ilities’) and their quality subfactors:

- Availability
- Interoperability
- Performance
  - Jitter, Response Time, Schedulability, and Throughput
- Portability
- Reliability
- Safety
- Security
- Usability
SAA Principles

The subsystem architects should know:

• What quality goals and requirements drove the development of their architectures.
• What architectural decisions they made.
• Why they made these decisions.
• Where these decisions are documented.

Because the subsystem architects should already have documented this information as a natural part of their architecting method, little new documentation should be necessary for the subsystem architects to make their cases to the subsystem assessment team.

The subsystem architects are responsible for making their own cases that their architectures adequately support their derived and allocated quality requirements.
SAA Phase Challenges

Architects may not have developed quality cases as a natural part of their architecting process:

- Architectural documentation typically not organized by quality factors.
- Quality case evidence is often buried in and scattered throughout massive amounts of architectural documentation.
- Architectural models (e.g., UML) often do not address support for quality requirements.

Architecture assessments may not be:

- Mandated by contract or development process
- Scheduled and funded

Managers feel schedule pressures do not allow time for assessment.
SAA Context

System Architecture Assessment Initiation Phase

<table>
<thead>
<tr>
<th>Prep.</th>
<th>Initial Meeting</th>
<th>Follow Through</th>
</tr>
</thead>
</table>

Subsystem 1 Architecture Assessment

Subsystem Requirements Review Phase

| Prep. | Rqmts. Meeting | Follow Through |

Subsystem Architecture Assessment Phase

| Prep. | Arch. Meeting | Follow Through |

Subsystem 2 Architecture Assessment

Subsystem Requirements Review Phase

| Prep. | Rqmts. Meeting | Follow Through |

Subsystem Architecture Assessment Phase

| Prep. | Arch. Meeting | Follow Through |

Subsystem N Architecture Assessment

Subsystem Requirements Review Phase

| Prep. | Rqmts. Meeting | Follow Through |

Subsystem Architecture Assessment Phase

| Prep. | Arch. Meeting | Follow Through |

System Architecture Assessment Summary Phase

| Prep. | Final Meeting | Follow Through |

Time (not to scale) →

© 2006 by Carnegie Mellon University
Version 0.2
QUASAR Method. - page 75
SAA Phase Overview

Subsystem Architecture Assessment Phase

Preparation Task
- Subsystem Architecture Assessment Checklist
- Subsystem Architecture Assessment Presentation Materials
- Subsystem Architecture Assessment Meeting Agenda

Meeting Task
- Subsystem Architecture Assessment Meeting Notes
- Subsystem Architecture Assessment Meeting Outbrief
-Subsystem Architecture Support Matrix

Follow-Through Task
- Updated Architecture Assessment Action Item List
- Architecture Assessment Procedure
- Architecture Assessment Training Materials

Assessment Team
Top-Level Architecture Team
Architecture Team
Subsystem Architecture Team

© 2006 by Carnegie Mellon University
Version 0.2
QUASAR Method. - page 76
SAA Preparation Task

Steps:
1. Subsystem Assessment Team Provides Assessment Checklist
2. Subsystem Architecture Team Gathers (Generates) and Makes Available Preparatory Materials:
   • Subsystem Architecture Overview
   • Updated Quality Requirements
   • Quality Cases including Arguments and Evidence
3. Subsystem Architecture Team Gathers (Generates) and Makes Available Presentation Materials
4. Subsystem Assessment Team:
   • Reads Materials
   • Generates RFIs and RFAs
5. Teams Collaborate to Organize Assessment Meeting (Attendees, Time, Location, Agenda, Invitation)
SAA Meeting Task

Steps:

1. Subsystem Architecture Team:
   • Introduces Subsystem Architecture
     (purpose, location, context, functions)
   • Reviews Architecturally-Significant Requirements
   • Introduces Subsystem Architecture
     (components, relationships, major decisions, trade-offs)
   • Present Quality Cases
     (claims, arguments, and evidence)

2. Subsystem Assessment Team:
   • Probes Architecture (quality case by quality case)
   • Manages Action Items
SAA Follow-Through Task

Steps:

1. Subsystem Assessment Team:
   - Develops Consensus
   - Produces, Reviews, and Presents Meeting Outbrief
   - Produces, Reviews, and Presents Subsystem Assessment Report
   - Manages Action Items
   - Captures Lessons Learned
   - Updates Assessment Method and Training Materials
SAA Roles and Responsibilities

The Subsystem Architecture Assessment Phase is performed by the following teams:

• Subsystem Architecture Team
• Subsystem Assessment Team
Subsystem Architecture Team

Responsibilities:

• Develop the architectural information to be presented during the meeting:
  - Architectural decisions and rationale
  - Supporting evidence

• Present this information to the subsystem assessment team

• Answer probing questions raised by the subsystem assessment team:
Subsystem Assessment Team

Responsibilities:

• Review supplied information prior to the subsystem architecture assessment meeting
• Assess the quality of the subsystem architecture:
  - Actively listen to the quality cases presented by the subsystem architecture team
  - Ask probing questions of Architects
SAA Discussion

Should the quality cases be developed as a:
  • Natural part of the architecting process?
  • Part of the assessment process?

How does the answer to the previous question affect the amount of time needed to prepare for the assessment meeting?

Which team has the most work to do during each task?

How should the development of the subsystem assessment report be divided up between members of the assessment team?
System Architecture Assessment Summary (SAAS)

System Architecture Assessment Initiation

Subsystem Requirements Review

Subsystem Architecture Assessment

repeat for each subsystem being assessed

no

done

yes

System Architecture Assessment Summary
SAAS Topics

SAAS Questions for the Attendees
SAAS Phase Objectives
SAAS Phase Principles
SAAS Phase Challenges
SAAS Phase Context
SAAS Phase Overview:
  • SAAS Preparation Task
  • SAAS Meeting Task
  • SAAS Follow-Through Task
SAAS Roles and Responsibilities
Discussion
SAAS Questions for the Attendees

How do you summarize the results of subsystem assessments at the system level?

Should the system architecture assessment summary phase be performed:
  • Once at the end?
  • On an ongoing rolling-wave basis?
SAAS Objectives

Collect previous Subsystem Architecture Assessment Results
Create System Architecture Assessment Summarize Results
Capture Method Lessons Learned
Update Assessment Method and Training Materials
SAAS Principles

All subsystems are not equally important.
All quality factors are not equally important for different subsystems.
It is probably better to concentrate on identifying problem/risk areas so that they can be fixed than to provide an overall summary assessment result.
SAAS Phase Challenges

How should subsystem findings be summarized without ending up comparing apples and oranges?

• Average Subsystem Architecture Quality
• Worst Subsystem Architecture Quality
• Union of Subsystem Architecture Qualities

Executive management may demand simplistic single number summary of system architecture.
SAAS Phase Overview

Preparation Task
- System Summary
- Subsystem Matrix
- System Summary Meeting Presentation Materials
- System Architecture Assessment Summary Meeting Agenda
- Subsystem Architecture Assessment Meeting Assessor Notes
- System Architecture Quality Assessment Summary Report
- Updated Architecture Assessment Action Item List
- Architecture Assessment Training Materials
- Architecture Assessment Procedure

Meeting Task
- System Assessment Team

Follow-Through Task
- System Architecture Team
SAAS Preparation Task

Steps:

1. System Assessment Team:
   - Collects Subsystem Architecture Assessment Results
   - Summarizes Subsystem Architecture Assessment Results
     - Develops Subsystem Architecture Support Matrix
   - Identifies Primary Stakeholders
   - Produces, Reviews, and Distributes:
     - System Architecture Quality Assessment Summary Report
     - Preparatory Materials
     - Meeting Agenda
   - Organizes Meeting
SAAS Meeting Task

Steps:

1. System Assessment Team:
   - Restates Assessment Objectives
   - Summarizes Assessment Method
   - Summarizes Quality of Subsystem Architectures
   - Summarizes Quality of System Architecture
   - Solicits Feedback
   - Captures Lessons Learned

2. System Architecture Team:
   - Captures Lessons Learned
SAAS Follow-Through Task

Steps:

1. System Assessment Team:
   • Updates and Distributes the System Architecture Assessment Summary Report
   • Manages Action Items
   • Updates Assessment Method and Training Materials

2. System Architecture Team:
   • Updates Architecture Method and Training Materials
SAAS Responsibilities

System Assessment Team:
  • Develop and Present System-Level Architecture Assessment Summary Results
  • Capture Lessons Learned
  • Update Assessment Method and Training Materials

System Architecture Team:
  • Validate Assessment Results
  • Capture Lessons Learned
  • Update Architecture Method and Training Materials

Management Team:
  • Manage Architectural Risks
SAAS Discussion

For a given quality factor, what is the best way to summarize the quality of the system architecture in terms of the quality of the architecture of the main subsystems?

• Average subsystem quality?
• Worst subsystem quality?
• Keep separate by listing individually?

What is the best way to summarize across all quality factors?

• Average value?
• Worst value?
• Keep separate by listing individually?
QUASAR Today and Tomorrow

Today:
• In-use on massive DoD Program
• Handbook published
• Provided as SEI Service

Future Plans:
• More Conference Tutorials
• QUASAR Training Materials and Classes
• QUASAR Articles
• Use and Validation on more Programs
• QUASAR Book
QUASAR Handbook

Intended Audiences:
• Acquisition Personnel
• Developers (Architects and Requirements Engineers)
• Subject Matter Experts (domain, specialty engineering)
• Consultants
• Trainers

Objectives:
• Completely Document the QUASAR method
• Enable Readers to start using QUASAR

Description:
• Very Complete
• Too comprehensive to be good first introduction
Questions?

For more information, contact:
Donald Firesmith
Acquisition Support Program
Software Engineering Institute
dgf@sei.cmu.edu
Systems Level Configuration Management
Contents of Course

- Introduction
  - Configuration Management Concepts
  - Configuration Management in Detail
  - Tailoring Configuration Management
  - Points to Remember
  - References / Suggested Reading
  - Questions / Answers / Discussion
  - Contact Information
Course Objectives

Provide students with an understanding of:

- Configuration Management (CM)
- Importance of CM
- Identification of Configuration Items (CIs)
- Baselines
- Controlling Changes to CIs
- Configuration Control Boards
- Classes of Changes
- Conducting Impact Assessments on Requested Changes
- Configuration Status Accounting
- Configuration Management Audits
- CM Responsibilities of Stakeholders
- CM Relationships between Acquirer and Supplier
Introduction (continued)

Why CM?

- CM ensures that the current configuration of items are known throughout their lifecycle
- CM ensures that changes to the configuration of evolving items are correct, controlled, managed, and documented
What is CM?

CM is a discipline applying technical and administrative direction and surveillance to:
- Identifying and documenting the physical, functional, and performance characteristics of items
- Baselining those characteristics
- Controlling changes to those characteristic
- Providing status on those characteristics
- Conducting audits on those characteristics

The CM tasks that produce these results are:
- Configuration Identification
- Configuration Control
- Configuration Status Accounting
- Configuration Management Audits
Application of CM

The CM concepts presented in this tutorial can be applied at the enterprise/systems/subsystem/program/project level to:
- Hardware
- Software
- Facilities
Introduction (continued)

Capability Maturity Model Integration (CMMI®)

The Software Engineering Institute’s CMMI® has a supporting Process Area that requires organizations that are developing systems to conduct a minimum set of CM tasks on the development and maintenance of products in order to achieve CMMI compliance.
CMMI (Continued)

Configuration Management Process Area (continued)

CMMI - Configuration Management

- **SG 1 Establish Baselines**
  - SP 1.1 Identify Configuration Items
  - SP 1.2 Establish a Configuration Management System
  - SP 1.3 Create or Release Baselines

- **SG 2 Track and Control Changes**
  - SP 2.1 Track Change Requests
  - SP 2.2 Control Configuration Items

- **SG 3 Establish Integrity**
  - SP 3.1 Establish Configuration Management Records
  - SP 3.2 Perform Configuration Audits

SG – Specific Goal
SP – Specific Practice

AI Florence MITRE
CMMI (completed)
Configuration Management Process Area (completed)

CMMI - Configuration Management

- GG 2 Institutionalize a Managed Process
  - GP 2.1 Establish an Organizational Policy
  - GP 2.2 Plan the Process
  - GP 2.3 Provide Resources
  - GP 2.4 Assign Responsibility
  - GP 2.5 Train People
  - GP 2.6 Manage Configurations
  - GP 2.7 Identify and Involve Relevant Stakeholders
  - GP 2.8 Monitor and Control the Process
  - GP 2.9 Objectively Evaluate Adherence
  - GP 2.10 Review Status with Higher Level Management
Introduction (continued)

Some Levels of CM

Enterprise CM

Supplier CM

Development CM
- Formal CM
- CI Characteristics
  - Physical
  - Function
  - Performance

Internal CM
- Design
- Implementation
- Code
- Test
- Process
- Documentation

Acquirer CM

Development CM
- Formal CM
- CI Characteristics
  - Physical
  - Function
  - Performance

Internal CM
- Business Cases
- Business Practices
- Budgets

Operational and Maintenance CM
*Could be a different contractor

Control Changes:
- Cost
- Schedule
- Interfaces

Control Changes:
- Whatever is necessary

*Operational and Maintenance CM
Introduction (continued)

Some Levels of CM (concluded)

◆ Enterprise CM
  – Covers all CM required for the entire enterprise (Acquirer/Supplier)

◆ Supplier CM - Formal CM
  – Development CM that concerns high level contractual issues such as specifications *(What shall be accomplished?)*

◆ Supplier CM - Internal CM
  – Development CM that concerns lower level contractual issues such as design, implementation, test, plans *(How is it accomplished?)*

◆ Acquirer CM - Development Formal CM
  – Acquirer CM that concerns high level contractual issues such as specifications *(What shall be accomplished?)*

◆ Acquirer CM - Internal CM
  – Acquirer CM that concerns internal business issues

◆ Operational & Maintenance CM
  – CM conducted after the system has been delivered and in operation
CM Focus in this Course

Introduction (Concluded)

This focus is chosen to serve as an example that can be applied, as appropriate, to other levels of CM and because this is one area in development that gets projects in trouble very quickly if not done properly.
Where are we?

- Introduction
- **Configuration Management Concepts**
  - Configuration Management in Detail
  - Tailoring Configuration Management
  - Points to Remember
  - References / Suggested Reading
  - Questions / Answers / Discussion
  - Contact Information
CM Concepts

System

Configuration Identification

Configuration Item

Configuration Control Board
  
Baseline

Configuration Control

Technical Review Board

Configuration Management Audits – Configuration Status Accounting
CM Concepts (continued)

System
◆ A composite of items (e.g., hardware, software, facilities, personnel, material, services, and techniques) required to perform a complete operational role

Configuration Identification
◆ The identified configuration of items such as hardware, software, and facilities within a system, and their physical, functional, and performance characteristics

Configuration Item
◆ An identified configuration of an item, or a portion of its parts, that is designated for change control
CM Concepts (continued)

**Configuration Item**

 Represents the characteristics of a Configuration Item

- **Functional and performance characteristics**
  - Rolls down hill at 10 mph
- **Physical characteristic**
CM Concepts (continued)

**Baseline**

- The approved and fixed (baselined) configuration of a CI at a specific time in its lifecycle that serves as a reference point for change control
  - CIs are used for visibility
  - Baselines are used for control
CM Concepts (continued)

Configuration Control

The systematic
- evaluation
- coordination
- approval or disapproval, and
- implementation

of changes to the physical, functional, and performance characteristics of a baselined CI
CM Concepts (continued)

Configuration Control Board (CCB)

◆ Establishes baselines for CIs
◆ Reviews and approves / disapproves / defers Change Requests to CIs
◆ Membership comprised of management and other stakeholders and supported by subject matter experts (SMEs)
  – Project Management
  – Systems Engineering
  – Software/Hardware Engineering
  – Test Engineering
  – Quality Assurance
  – Configuration Management
◆ Chaired by the program / project manager or designee
CM Concepts (continued)

Technical Review Board (TRB)

- Provides technical and programmatic support to the CCB
  - Conducts impact assessment on change requests (CRs) to baselined CIs
  - Makes approval / disapproval recommendations to the CCB
- Membership comprised of program / project personnel and subject matter experts
- Chaired by a technical manager
CM Concepts (concluded)

Configuration Management Audits

- Audits are conducted on CM tasks by the CM organization and Quality Assurance to ensure that CM is being executed as described in CM process documentation.

- At the end of development and prior to delivery, audits are conducted for the Acquirer to:
  - Ensure that all products comply with their requirements.
  - Ensure that all products comply with their design documents such as the software design, hardware design and facilities design documents.
CM Concepts (continued)

Configuration Status Accounting (CSA)

- CSA is performed to gather, correlate, maintain and provide status on controlled products (CIs), and on CM tasks.

<table>
<thead>
<tr>
<th>Specifications</th>
<th>Products (CIs)</th>
<th>CM Tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Configuration Identification
- Configuration Control
- Configuration Audits
- Configuration Status Accounting
Where are we?

- Introduction
- Configuration Management Concepts
- Configuration Management in Detail
- Tailoring Configuration Management
- Points to Remember
- References / Suggested Reading
- Questions / Answers / Discussion
- Contact Information
Configuration Management in Detail

This section will cover the following:

- Configuration Management Planning
- Configuration Identification
- Configuration Control
- Configuration Status Accounting
- Configuration Management Audits
CM Planning

- CM planning is essential if CM is to be effectively applied throughout the lifecycle of products.
- When conducting CM planning for a particular program / project, the size, type, and scope of the applications and the program / project needs to be accounted for in order to provide the correct amount of CM.
CM Planning (continued)

Planning Activities Are:
- Identifying CM Tasks
- Defining CM Roles and Responsibilities
- Selecting CM Tools
- Determining CM Resources
- Determining CM Training
- Defining CM Metrics
- Developing the CM Plan which is the output of planning
CM Planning (continued)

Identifying CM Tasks

- Configuration Identification
- Configuration Control
- Configuration Audits
- Configuration Status Accounting
CM Planning (continued)

Defining CM Roles and Responsibilities

- CM is **NOT** solely the responsibility of the CM organization
- CM involves all stakeholders of the program / project
- All organizations involved with the engineering and development of program / projects products have CM roles and responsibilities
CM Planning (continued)

Defining CM Roles and Responsibilities (continued)

◆ Configuration Management Manager
  – Primary responsibility for the development of the CM plan
  – Responsible for overseeing tasks assigned to the CM Organization and ensuring that they are performed
  – Submits CM plans for approval
  – Serves on the CCB (provides scribe)
  – Conducts impact assessments on CRs
CM Planning (continued)
Defining CM Roles and Responsibilities (continued)

Configuration Management Practitioners

- Primary responsibility for developing, implementing and maintaining CM plans, processes, procedures, and tools
- Responsible for CM Repository*
- Prepare agendas and minutes for CCB meetings
- Administration of CRs
- Tracks the implementation of approved CRs
- Conducts CM Process audits
- Responsible for CM Status Accounting

*CM Repository stores CM documentation, CM records, CM artifacts, etc.
CM Planning (continued)
Defining CM Roles and Responsibilities (continued)

◆ **Program/Project Manager**
  – Authorizes CM plans
  – Ensures adequate CM resources are provided
  – Enforces CM tasks
  – Chairs the Configuration Control Board (CCB)

◆ **Quality Assurance**
  – Audits CM activities to ensure that CM tasks are conducted in accordance with documented plans and processes
  – Conducts impact assessments on CRs
  – Serves on the CCB
CM Planning (continued)
Defining CM Roles and Responsibilities (continued)

◆ **Systems Engineering**
  - Primary responsibility for definition and identification of system level CIs
  - Conducts impact assessments on CRs
  - Conducts product audits at end of development
  - Serves on the CCB

◆ **Hardware / Software Engineering**
  - Primary responsibility for definition, identification and implementation of hardware / software CIs
  - Conducts impact assessments on CRs
  - Conducts product audits at end of development
  - Serves on the CCB
CM Planning (continued)
Defining CM Roles and Responsibilities (concluded)

◆ Test Engineering
  – Responsible for testing CIs
  – Conducts impact assessments on CRs
  – Provide test-related CM artifacts for the CM repository
  – Conducts product audits at end of development
  – Serves on the CCB
**CM Planning (continued)**

**Defining CM Roles and Responsibilities (concluded)**

<table>
<thead>
<tr>
<th>Organizations</th>
<th>CM Planning</th>
<th>Configuration Identification</th>
<th>Configuration Control</th>
<th>Configuration Status Accounting</th>
<th>Configuration Management Audits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project Manager</td>
<td>Approves</td>
<td>Supports</td>
<td>Approves Changes</td>
<td>Receives Reports</td>
<td>Supports</td>
</tr>
<tr>
<td>Configuration Management Organization</td>
<td>Conducts</td>
<td>Facilitates</td>
<td>Facilitates process</td>
<td>Conducts</td>
<td>Facilitates Conducts</td>
</tr>
<tr>
<td>Quality Assurance</td>
<td>Audits Process Artifacts</td>
<td>Audits Process Artifacts</td>
<td>Audits Process Artifacts</td>
<td>Receives &amp; Audits Reports</td>
<td>Witnesses Conducts</td>
</tr>
<tr>
<td>Systems Engineering</td>
<td>Supports</td>
<td>Conducts</td>
<td>Supports</td>
<td>Receives Reports</td>
<td>Conducts Product Audits</td>
</tr>
<tr>
<td>HW/SW Engineering</td>
<td>Supports</td>
<td>Conducts</td>
<td>Supports</td>
<td>Receives Reports</td>
<td>Conducts Product Audits</td>
</tr>
<tr>
<td>Test Engineering</td>
<td>Supports</td>
<td>Supports</td>
<td>Supports</td>
<td>Receives Reports</td>
<td>Conducts Product Audits</td>
</tr>
</tbody>
</table>

* CM facilitates Product Audits and conducts other CM audits
** QA may also facilitate Product Audits and conduct other CM audits
Support involves providing information and subject matter expertise, and reviewing artifacts

Al Florence MITRE
CM Planning (continued)

Selecting CM Tools

CM tools should be used to:

- Store CM documentation and artifacts
- Control versions
- Track and status CRs
- Support administration and communication
  - Create documents and reports
  - Develop presentations
  - Produce schedules
  - Collect measurements
  - Conduct analysis / create metrics
CM Planning (continued)

Determining CM Resources

◆ **CM resources are comprised of:**
  - CM personnel
  - Facilities
  - Funding
  - Equipment
  - Tools
  - Supplies
  - Administrative support
CM Planning (continued)

Determining CM Training

- CM training may include:
  - Project management level CM orientation
    - CM roles and responsibilities
  - Program / project staff CM orientation and training
    - Orientation on CM roles and responsibilities
    - Training on CM activities
  - CM practitioner training
    - CM roles and responsibilities
    - In-depth CM training on CM activities
    - Use of CM tools and CM repository
Determining CM Metrics

Measurements are collected during the execution of the CM activities. The following are some typical CM measurements:

- **Baselines**
  - Number of items pending baselining
  - Number of actual items baselined

- **Change Requests**
  - Number of CRs approved
  - Number of CRs implemented

- **For each CR**
  - Date approved
  - Planned date closed
  - Actual date closed
CM Planning (continued)

Determining CM Metrics

◆ The measurements analyzed which result in metrics which can be shown in charts, graphs, tables, etc.

◆ Metrics can be used to:
  – Provide status to management on the CM activities, products and services
  – Determine behavior of the CM process used to conduct the activities that produce products and services
  – Make management decisions and corrections to bring products and activities under control when required
  – Identify areas where the CM or engineering process is unstable, which may lead to process improvement
CM Planning (continued)

Metrics Example

Change Requests as of November 4, 2003

<table>
<thead>
<tr>
<th>Number Approved</th>
<th>Number Implemented</th>
<th>Number Not Implemented</th>
<th>Number Missed Due Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>35</td>
<td>20</td>
<td>10</td>
<td>4</td>
</tr>
</tbody>
</table>

![Bar chart showing change requests status as of November 4, 2003]
CM Planning (continued)

Developing the CM Plan

◆ The purpose of the CM Plan is to describe the processes, products, and organizational responsibilities required to implement effective CM functions on programs / projects

◆ The Plan presents CM as it is applied to a particular program / project and tailors the CM approach appropriately to the scope of the application
  - A large application needs BIG CM
  - A small application needs small CM

◆ More on tailoring CM will be presented later

The CM Plan should not include detailed CM processes that require frequent update
CM Planning (concluded)

Developing the CM Plan (concluded)

The Plan contains the following:

- **Introduction of the plan’s purpose, scope, and contents**
- **An overview of the CM tasks**
  - Configuration Identification
  - Configuration Control
  - Configuration Audits
  - Configuration Status Accounting
- **Organizational CM roles and responsibilities**
- **CM risk management**
- **CM resources**
- **CM metrics**
- **High level milestone and schedules relating to CM**
Configuration Management in Detail

- Configuration Management Planning
- Configuration Identification
  - Configuration Items
  - Baselines
- Configuration Control
- Configuration Status Accounting
- Configuration Management Audits
Configuration Identification

- Configuration Identification is established in the form of documentation of items that becomes more detailed as development proceeds.
- It is important to assign unique identifiers to items.
  - Supports version identification and control.
Configuration Identification (continued)

- Three levels of Configuration Identification are established
  - Functional Configuration Identification (FCI)
  - Allocated Configuration Identification (ACI)
  - Physical Configuration Identification (PCI)

Conceptual Systems Requirements

- Hardware Software Facilities Requirements

Design Implementation Test Operation

Lifecycle Phases

AI Florence MITRE
The identified system and system items and their physical, functional, and performance characteristics which are documented in a System CI Specification.
Allocated Configuration Identification (ACI)—\(^1,^2\)

Later in development the physical, functional, and performance characteristics of the system are allocated to lower level entities: software, hardware, facilities, and documented as CI Specifications.
Configuration Identification (continued)

Physical Configuration Identification

Physical Configuration Identification (PCI)\textsuperscript{1, 2}

Finally, the products of the developed system: software, hardware, facilities are defined in a series of Product CI Specifications that describe the as-built system.

\textbf{As-built System}

Product CI Specifications
Configuration Identification (concluded)

- Functional Configuration Identification (FCI)
  - System CI Specification
  - Conceptual Phase
- Allocated Configuration Identification (ACI)
  - Software CI Specifications
  - Hardware CI Specifications
  - Facilities CI Specifications
  - Development Phases
- Physical Configuration Identification (PCI)
  - Designed/Built/Tested Entities
  - Design/Built/Tested Entities
  - Design/Built/Tested Entities
  - Operational Phase
- As-Built Products

AI Florence MITRE
Configuration Identification and Configuration Items

- Configuration Identification is an activity that identifies items and their characteristics: physical, functional, and performance.
- Not all items that are identified need be controlled at the same level of rigor.
- Configuration Items are selected for formal change control from items identified.

### Configuration Identification - Software

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>*Operating System</td>
<td>*Network Software</td>
<td>**Navigation Software CI</td>
<td>**Communication Software CI</td>
</tr>
<tr>
<td>**Test Software CI</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*These are commercial products not subject to change – In operation (production) everything is under CM control.**

**Applications software in development that is subject to change.
Baselines

Baselines are established at strategic points in a system lifecycle. Three baselines may be defined:

- Functional Baseline (FBL)
- Allocated Baseline (ABL)
- Product Baseline (PBL)

Lifecycle Phases

Systems Requirements

Hardware Software Facilities Requirements

Design Implementation Test Operation

FCI - CI ACI - CIs PCI - CIs

FBL ABLs PBLs
Baselines (continued)

**Functional Baseline**

**Functional Baseline (FBL)—Established for the CI of the System Functional Configuration Identification when System Requirements Phase of the system lifecycle is completed**\(^1,2\)

- System CI Specification
- FBL
- Under Formal Change Control by the CCB
- Time
- End of Systems Requirements Phase
Baselines (continued)

Allocated Baseline

Allocated Baseline (ABL)—\(^1, 2\) Established for the CIs of the Allocated Configuration Identification later in development for software, hardware, facilities, etc.

Software CI Specification

Hardware CI Specification

Facilities CI Specification

End of Software/Hardware/Facilities Requirements Phase

Under Formal Change Control by the CCB

Time

ABL
Baselines (continued)

Product Baseline

Product Baseline (PBL)\(^1, 2\) Established for the developed products (CIs) of the Physical Configuration Identification at the end of development

- Under Formal Change Control by the CCB
- During Operation

End of Development Prior to Delivery to Customer
Baselines (continued)

Functional Baselines (FBL) for:
- System CI

Allocated Baselines (ABL) for:
- Hardware CIs
- Software CIs
- Facilities CIs

Product Baselines (PBL) for:
- Hardware CIs
- Software CIs
- Facilities CIs

Conceptual Phase

Development Phases

Operational Phase

As-Built Products
Baselines (concluded)

It gets more complex:

- As development progresses CIs evolve and include more detail:
  - Initially the CIs are represented as requirements documented in CI Requirements Specifications
  - Later the CIs are represented in:
    - Design documents
    - Test plans
    - Code (for software)
    - Test procedures
    - Test results
  - During development only CIs that have achieved the Functional Baseline and the Allocated Baseline for the CI Specifications are designated for formal CCB control*

* As described for this presentation and as reflected in references 1 and 2.
Example

A NASA spacecraft (Galileo) orbits Jupiter and releases a probe to conduct scientific experiments on the planet’s atmosphere.
Example (continued)

Functional Configuration Identification

1. Identify 3 configuration Items of one subsystem – earth station, spacecraft, probe
2. List the some physical, functional, and performance characteristics of one item
3. Allocate these characteristics to software, hardware, systems, and facilities as appropriate
4. Write these allocations as “shall” requirements
Example (continued)

- **Jupiter**
  - A gas planet
  - No terrestrial surface
  - High atmospheric pressures
  - Compresses to a hard state at surface

- **Spacecraft**
  - 45 minute one way communication delay at speed of light to/from earth/spacecraft

- **Probe**
  - Autonomous control
  - One way communications from probe to spacecraft
  - 40 minute science data collection life
  - Implodes due to high atmospheric pressures
Example (continued)

PROBE

Configuration Items:
1. Parachute
2. Heat Shield
3. Science Instruments

Heat Shield characteristics:

A. Physical:
   A.1 Three feet in diameter
   A.2 Half inch thick graphite
   A.3 Perfect circular and spherical curvature shape

B. Functional:
   B.1 Protect probe from over heating

C. Performance:
   C.1 Jettisoned 48 hours 20 minutes after separation from spacecraft
   C.2 Jettisoned within one second of jettison command
   C.3 Withstands temperatures up to 500 degrees Fahrenheit
Example (continued)

Probe Heat Shield

Heat Shield Allocations:

A. Physical:
   A.1 Three feet in diameter
   A.2 Half inch thick graphite
   A.3 Perfect circular and spherical curvature shape

B. Functional
   B.1 Protect probe from over heating

C. Performance
   C.1 Jettisoned 48 hours 20 minutes after separation from spacecraft
   C.2 Jettisoned within one second of jettison command
   C.3 Withstands temperatures up to 500 degrees Fahrenheit

Hardware (HW)
Hardware
System
Software/HW
Hardware
Example (completed)

Probe Heat Shield

Allocated Shall Requirements:

HW1 Heat Shield physical hardware requirements
  HW1.1 The heat shield shall be three feet in diameter.
  HW1.2 The heat shield shall be constructed of half inch thick graphite.
  HW1.3 The heat shield shall be of a perfect circular and spherical curvature shape.

Sys1 Heat Shield functional system requirements
  Sys1.1 The heat shield shall protect the probe from over heating.

SW1 Heat Shield software performance requirements (Also Hardware)
  SW1.1 The heat shield shall be jettisoned 48 hours 20 minutes after separation from spacecraft.
  SW1.2 The heat shield shall be jettisoned within one second of jettison command.

HW2 Heat Shield hardware performance requirements
  HW2.1 The heat shield shall withstand temperatures up to 500 degrees Fahrenheit.
Configuration Management in Detail

- Configuration Management Planning
- Configuration Identification
- **Configuration Control**
  - CCB and TRB
- Configuration Management Audits
- Configuration Status Accounting
CM Concepts in Detail (continued)

Configuration Control

Configuration Item

- Functional and performance characteristic
- Physical characteristic

Rolls down hill at 10 mph

Constraint
- 3% Grade
- Gravel

Change Request

If not we might get something really dumb or suffer a catastrophic failure

Configuration Item

- Less than 3 mph wind

Slides down hill at 15 mph

Need to control the configuration of physical, functional, and performance characteristic

Al Florence MITRE
Configuration Control (continued)

How are Changes Accomplished?

- **Request Change**
  - Someone requests a change to a CI using a CR form

- **Evaluate Change**
  - The TRB conducts an impact assessment to ensure that all stakeholders evaluate the impact against their interests

- **Approve Change**
  - The CCB approves, disapproves or defers the CR

- **Implement Change**
  - The change is implemented in all affected items

- **Track Changes**
  - Changes are audited to verify that they are implemented as approved and tracked against the change schedule
CM Concepts (concluded)

CR Example

<table>
<thead>
<tr>
<th>CR #</th>
<th>Date: 12/4/2003</th>
<th>Requestor: ET</th>
<th>Class: I □ II □</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Problem:</strong></td>
<td>A requirement to deploy the probe’s parachute does not exist</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Change:</strong></td>
<td>Add the following requirement: The probe’s parachute shall be deployed 10 seconds after the heat shield has been jettisoned</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Impacts:</strong></td>
<td>Enter figures for cost and schedule and list affected interfaces or “None” and attach impact assessments</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Systems:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hardware:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Software:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Configuration Management:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Quality Assurance:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Contracts:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Other [Specify]:</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Approve:**
- TRB Date: 
- CCB Date: 
  - Chair: 

**Disapprove:**
- TRB Date: 
- CCB Date: 
  - Chair: 

**Assignee:**
- Due Date:
Configuration Control (continued)

Change Flow

- Request Change: Supplier or Acquirer
- Evaluate Change: TRB
- Approve Change: CCB
- Implement Change: Owner of item
- Track Change: CM staff and owner of item
Impact Assessments

Impact assessments need to be conducted by all stakeholders:
- Systems
- Hardware
- Software
- Test
- Configuration Management
- Quality Assurance
- Contracts
- Others

On CI characteristics:
- Physical
- Functional
- Performance

Against their interests:
- Cost
- Schedule
- Interface
Configuration Control (continued)

Classification of Changes

At least two types of changes can be defined:

1, 2 Class I—affects the Acquirer’s interest in one or more of these factors:
  - Physical characteristics
  - Functional capability
  - Performance
  - External interfaces
  - Cost
  - Schedule

Supplier must submit change to the Acquirer for approval before implementation (Based on Thresholds)
1, 2 Class II
- Does not affect any of the class I factors
- Affects changes such as:
  - Spelling or typographical errors
  - Addition of clarifying comments
  - Changes that do not affect external interfaces, change functionality or degrade performance

Supplier may implement it without Acquirer’s approval but must inform Acquirer of change
CM Concepts in Detail (continued)

CCB and TRB

◆ CCB is a formal board dealing with contractual items such as requirements specifications

◆ CCB membership consists of senior and program management
  – Very busy, and “$$” to deal with lower-level items

◆ TRBs are less formal and deal with internal control of items such as design, implementation, and test

◆ TRBs act as a winnowing agent on items that should not go to the CCB

◆ TRBs conduct impact assessments on CRs and make recommendation to the CCB of approval or rejection

◆ TRB membership consist of program technical management and subject matter experts, “$$” that provides technical support to the CCB
CCB and TRB (continued)  

**CCB and TRB Hierarchy**

```
Acquirer
CCB

Supplier Program
CCB

Supplier Project A
CCB

Supplier Project N
CCB

TRB

Acquirer (Customer)

Supplier (Contractor)
```

Al Florence  MITRE
CCB and TRB (continued)

CCB and TRB Change Flow

Change Request

- Project TRB
  - Class I
    - Incorporate
      - Impact Assessment & CCB Recommendation
        - Project CCB
          - Affects Higher CI
            - Yes
              - Approve
            - No
              - Can reject
            - Done
          - Next Slide
          - Can reject
          - Incorporate
        - Next higher level TRB needs to verify the assessments and the recommendation
      - Affects Higher CI
        - Yes
          - Acquirer TRB
        - No
          - Done
      - Program TRB
        - Incorporate
        - Yes
          - Done
        - No
          - Affects Higher CI
            - Yes
              - Incorporate
            - No
              - Done

CCB and TRB Change Flow (continued)
CCB and TRB (concluded)

CCB and TRB Change Flow (concluded)

Prior Slide

Program CCB

Affects Higher CI

Acquirer CCB

Can reject

Approve

Incorporate

Project CCB

Can reject

Approve

Incorporate

Project CCB

Program CCB

Incorporate

Incorporate

Incorporate

Based on thresholds

Can reject

No

Can reject

Approve

Incorporate

Project CCB

Incorporate

Incorporate

Incorporate

Can reject

Approve

Incorporate

Project CCB

Incorporate

Incorporate

Incorporate

Can reject

Approve

Incorporate

Project CCB

Incorporate

Incorporate

Incorporate

Can reject

Approve

Incorporate

Project CCB

Incorporate

Incorporate

Incorporate

Can reject

Approve

Incorporate

Project CCB

Incorporate

Incorporate

Incorporate

Can reject

Approve

Incorporate

Project CCB

Incorporate

Incorporate

Incorporate

Can reject

Approve

Incorporate

Project CCB

Incorporate

Incorporate

Incorporate

Can reject

Approve

Incorporate

Project CCB

Incorporate

Incorporate

Incorporate

Can reject
Configuration Management in Detail

- Configuration Identification
- Configuration Items
- Baselines
- Configuration Control
- Configuration Management Audits
- Configuration Status Accounting
CM Audits

- Functional Configuration Audits (FCA) and Physical Configuration Audits (PCA) are conducted by Engineering and facilitated by CM and/or QA

- Other audits conducted by QA and CM may include:
  - Audits of CM Repository that contains CM records, documentation, processes, procedures, artifacts, etc.
  - Audits of Program/Project organizations to ensure CM process is being followed
  - Audits of status of approved CRs
  - Audits to ensure that CIs are consistent with CM records
Functional Configuration Audit (FCA)

A formal examination of test results of the as-built functional configuration of CIs, prior to acceptance, to verify that the CIs have satisfied their specified requirements \(^1,^2\)

This audit is conducted by the Supplier for the Acquirer and attended by

- Management
- System Engineering
- Hardware / Software Engineering
- Test Engineering
- QA and CM
- Contracts

of both the Acquirer and Supplier
CM Audits (continued)
FCA (completed)

Functional
- Requirements Specifications
- Requirements Traceability
- Test Plans
- Test Scenarios

Testing
- Products
- Tests

Test Results

Functional Configuration Audit
Verify that the CIs have satisfied their specified requirements
Supplier Acquirer

Physical Configuration Audit
CM Audits (continued)

Physical Configuration Audit (PCA)

- A formal examination of the as-built physical configuration of CI products against their design documentation $^1,^2$
- This establishes the Product Baseline
- This audit is conducted by the Supplier for the Acquirer and attended by
  - Management
  - System Engineering
  - Hardware / Software Engineering
  - Test Engineering
  - QA and CM
  - Contracts
  of both the Acquirer and Supplier
CM Audits (completed)
PCA (completed)

Supplier As-Built Products:
- Design Documentation
- Code
- Hardware
- Etc.

Physical Configuration Audit
Examination of the “as-built” configuration of CIs against their documentation

Physical

Implementation

Design Documentation

Supplier Acquirer

Outputs

Product Baselines

Inputs

AI Florence MITRE
Configuration Management in Detail

- Configuration Management Planning
- Configuration Identification
- Configuration Control
- Configuration Management Audits
- Configuration Status Accounting
The Configuration Status Accounting (CSA) task gathers, correlates, maintains, and provides status on CM controlled products and CM tasks.

- Provides the means for reporting status on:
  - Configurations:
    - FCI
    - ACI
    - PCI
  - Baselines:
    - FBL
    - ABL
    - PBL
  - Other:
    - CM metrics
    - CM activities
    - CM Audits

- Conducted by both the Supplier and the Acquirer
Configuration Status Accounting (concluded)

Supplier

Configuration Status Accounting Reports produced by the CM organization

Management and Staff

Acquirer

Monthly Reports

Program Management Reviews

Milestone Reviews
Where are we?

- Introduction
- Configuration Management Concepts
- Configuration Management in Detail
- Tailoring Configuration Management
- Points to Remember
- References / Suggested Reading
- Questions / Answers / Discussion
- Contact Information
Tailoring Configuration Management

The shoe “gotta” fit to be comfortable

AI Florence MITRE
Tailoring CM (continued)

◆ CM can be very dangerous if under or over applied
  – Too much CM can stifle projects with bureaucracy
  – Too little CM will result in the projects getting out of control

◆ No CM will result in late deliveries, cost overruns, poor reliability, and even total failure

◆ CM needs to be tailored and appropriately applied to the scope of the application; the following are some factors to consider when tailoring:
  – Cost
  – Schedule
  – Function
  – Performance
  – Safety
  – Security
  – Criticality
  – Reliability
  – Size of Application
  – Number of Suppliers
  – Relationship of Supplier / Acquirer
  – Number of Staff
Tailoring CM (concluded)

- For large, complex, critical projects, CM needs to be applied to its fullest formal extent.
- For smaller, less complex projects, CM may need to be tailored by analyzing the tailoring factors.
  - For example, a TRB may not be necessary.
- For small, non-complex projects (6 or fewer staff members, $500,000, 6 months) a formal CCB may not even be necessary.
- The important point is to apply the concepts and principles of CM as appropriate and necessary.
Where are we?

- Introduction
- Configuration Management Concepts
- Configuration Management in Detail
- Tailoring Configuration Management
- Points to Remember
- References / Suggested Reading
- Questions / Answers / Discussion
- Contact Information
Points to Remember

- CM is important to ensure that current configurations of items are known throughout their lifecycle and that changes to those configurations are managed and controlled.
- CM starts early in the development lifecycle and continues until the system is removed from operation.
- Configuration items are baselined at a specific time in their lifecycle as a reference point for change control.
Points to Remember

- Impact assessments must be conducted on CRs against function, cost, schedule, interface by all affected entities
- CM needs to be tailored and appropriately applied to the scope of the application
- CM relationships and responsibilities between the Acquirer and Supplier must be understood and adhered to
- All organizations have CM roles and responsibilities which need to be appropriately applied if CM is to be successful
References / Suggested Reading


4. *CM BoK – Configuration Management Body of Knowledge*. [www.cmcrossroads.com/cgi-bin/cmwiki/bin/view.cgi/CM/CMBok](http://www.cmcrossroads.com/cgi-bin/cmwiki/bin/view.cgi/CM/CMBok), CM Crossroads, CM Community Forums
References / Suggested Reading


Questions / Answers
Discussion
Contact Information

Al Florence
Florence@MITRE.org
(703) 983-7476
GRAND SYSTEMS
DEVELOPMENT TRAINING
PROGRAM

VERSION 10.1

The union of system engineering,
domain engineering, functional management,
and program management
for the greater good of the enterprise
and customer base.

VOLUME 2R
AN EFFECTIVE SPECIFICATION
DEVELOPMENT ALGORITHM

Presented By
Jeffrey O. Grady

6015 Charae Street
San Diego, California 92122
(858) 458-0121
(858) 456-0867 Fax

jgrady@ucsd.edu or jeff@jogse.com
http://www.jogse.com

Copyright 2006

No part of this manual may be scanned or reproduced in any
form without permission in writing from the author.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>PARAGRAPH</th>
<th>TITLE</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Specification Templates and DIDs</td>
<td>2</td>
</tr>
<tr>
<td>1.1</td>
<td>Enterprise Engineering Work</td>
<td>2</td>
</tr>
<tr>
<td>1.2</td>
<td>System Engineering Generic Work</td>
<td>3</td>
</tr>
<tr>
<td>1.3</td>
<td>Proposal Work That Prepares for Program Execution</td>
<td>4</td>
</tr>
<tr>
<td>1.4</td>
<td>Work Subsequent to Contract Award</td>
<td>6</td>
</tr>
<tr>
<td>1.5</td>
<td>The Preferred Templates</td>
<td>7</td>
</tr>
<tr>
<td>1.6</td>
<td>Modeling Work Product Capture Document</td>
<td>9</td>
</tr>
<tr>
<td>2</td>
<td>Structured Analysis</td>
<td>9</td>
</tr>
<tr>
<td>2.1</td>
<td>Traditional Structured Analysis</td>
<td>9</td>
</tr>
<tr>
<td>2.1.1</td>
<td>A System Defined</td>
<td>11</td>
</tr>
<tr>
<td>2.1.2</td>
<td>The System Environment</td>
<td>11</td>
</tr>
<tr>
<td>2.1.3</td>
<td>System Functionality</td>
<td>12</td>
</tr>
<tr>
<td>2.1.4</td>
<td>Performance Requirements Derivation and Allocation</td>
<td>15</td>
</tr>
<tr>
<td>2.1.5</td>
<td>Product Entity Structure</td>
<td>15</td>
</tr>
<tr>
<td>2.1.6</td>
<td>Allocation Pacing Alternatives</td>
<td>17</td>
</tr>
<tr>
<td>2.1.7</td>
<td>System Relations</td>
<td>18</td>
</tr>
<tr>
<td>2.1.8</td>
<td>Environmental Relation Algorithm</td>
<td>20</td>
</tr>
<tr>
<td>2.1.8.1</td>
<td>System Environmental Relations</td>
<td>20</td>
</tr>
<tr>
<td>2.1.8.2</td>
<td>End Item Service Use Profile</td>
<td>20</td>
</tr>
<tr>
<td>2.1.8.3</td>
<td>Component Environmental Relations</td>
<td>21</td>
</tr>
<tr>
<td>2.1.8.4</td>
<td>Environmental Impact</td>
<td>22</td>
</tr>
<tr>
<td>2.1.9</td>
<td>Specialty Engineering and RAS Complete</td>
<td>22</td>
</tr>
<tr>
<td>2.1.10</td>
<td>RAS-Complete in Table Form</td>
<td>24</td>
</tr>
<tr>
<td>2.1.11</td>
<td>Traditional Structured Analysis Summary</td>
<td>25</td>
</tr>
<tr>
<td>2.1.12</td>
<td>SDD Content and Format</td>
<td>26</td>
</tr>
<tr>
<td>2.1.12.1</td>
<td>Document Main Body</td>
<td>27</td>
</tr>
<tr>
<td>2.1.12.2</td>
<td>Appendix A, Functional Analysis</td>
<td>27</td>
</tr>
<tr>
<td>2.1.12.3</td>
<td>Appendix B, System Environment Analysis</td>
<td>27</td>
</tr>
<tr>
<td>2.1.12.4</td>
<td>Appendix C, System Architecture Analysis</td>
<td>28</td>
</tr>
<tr>
<td>2.1.12.5</td>
<td>Appendix D, System Interface Analysis</td>
<td>28</td>
</tr>
<tr>
<td>2.1.12.6</td>
<td>Appendix E, Specialty Engineering Definition Analysis</td>
<td>28</td>
</tr>
<tr>
<td>2.1.12.7</td>
<td>Appendix F, System Process Analysis</td>
<td>28</td>
</tr>
<tr>
<td>2.1.12.8</td>
<td>Appendix G, Requirements Analysis Sheet</td>
<td>28</td>
</tr>
<tr>
<td>2.1.13</td>
<td>Team Activity During Requirements Work</td>
<td>29</td>
</tr>
<tr>
<td>2.2</td>
<td>UML</td>
<td>30</td>
</tr>
<tr>
<td>2.2.1</td>
<td>Entry Analysis and Overview</td>
<td>30</td>
</tr>
<tr>
<td>2.2.2</td>
<td>The Connection Between Modeling Artifacts, Specification Content, and Product Entities</td>
<td>32</td>
</tr>
<tr>
<td>2.2.3</td>
<td>Dynamic Modeling Artifacts Explained</td>
<td>35</td>
</tr>
<tr>
<td>2.2.3.1</td>
<td>Sequence Diagram</td>
<td>35</td>
</tr>
<tr>
<td>2.2.3.2</td>
<td>Communication Diagram</td>
<td>37</td>
</tr>
<tr>
<td>2.2.3.3</td>
<td>Activity Diagram</td>
<td>37</td>
</tr>
</tbody>
</table>
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>PARAGRAPH</th>
<th>TITLE</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.2.3.4</td>
<td>State Diagram</td>
<td>38</td>
</tr>
<tr>
<td>2.2.4</td>
<td>Structural Analysis</td>
<td>39</td>
</tr>
<tr>
<td>2.2.4.1</td>
<td>The Class</td>
<td>40</td>
</tr>
<tr>
<td>2.2.4.2</td>
<td>Class Relationships</td>
<td>41</td>
</tr>
<tr>
<td>2.2.4.3</td>
<td>Messages</td>
<td>42</td>
</tr>
<tr>
<td>2.2.5</td>
<td>Related Analyses</td>
<td>42</td>
</tr>
<tr>
<td>2.2.5.1</td>
<td>Specialty Engineering</td>
<td>42</td>
</tr>
<tr>
<td>2.2.5.2</td>
<td>Environmental Requirements</td>
<td>42</td>
</tr>
<tr>
<td>2.2.6</td>
<td>Specification Structure</td>
<td>42</td>
</tr>
<tr>
<td>2.2.7</td>
<td>Software Requirements Close-out</td>
<td>44</td>
</tr>
<tr>
<td>2.3</td>
<td>Opening the Analysis With DoDAF</td>
<td>45</td>
</tr>
<tr>
<td>2.4</td>
<td>Integrated Modeling</td>
<td>47</td>
</tr>
<tr>
<td>3.1</td>
<td>Summary of Team Activity During Requirements Work</td>
<td>51</td>
</tr>
<tr>
<td>3.2</td>
<td>Requirements Tools Base</td>
<td>52</td>
</tr>
<tr>
<td>3.3</td>
<td>Recommended Specification Responsibility Pattern</td>
<td>53</td>
</tr>
<tr>
<td>3.4</td>
<td>Requirements Risk Management</td>
<td>54</td>
</tr>
<tr>
<td>3.4.1</td>
<td>Requirements Validation</td>
<td>54</td>
</tr>
<tr>
<td>3.4.2</td>
<td>Margins and Budgets</td>
<td>55</td>
</tr>
<tr>
<td>3.4.3</td>
<td>Risk Tracking</td>
<td>55</td>
</tr>
<tr>
<td>3.5</td>
<td>Verification Requirements</td>
<td>58</td>
</tr>
<tr>
<td>3.6</td>
<td>Specification Review and Approval</td>
<td>59</td>
</tr>
<tr>
<td>A</td>
<td>APPENDIX A, PRESENTATION MATERIALS</td>
<td>A-i</td>
</tr>
<tr>
<td>B</td>
<td>APPENDIX B, SPECIFICATION DATA ITEM DESCRIPTIONS</td>
<td>B-i</td>
</tr>
<tr>
<td></td>
<td>JOGSE System Specification Data Item Description</td>
<td>B-1-1</td>
</tr>
<tr>
<td></td>
<td>JOGSE Hardware Item Performance Specification Data Item</td>
<td>B-2-1</td>
</tr>
<tr>
<td></td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td></td>
<td>JOGSE Software Requirements Specification Data Item</td>
<td>B-3-1</td>
</tr>
</tbody>
</table>

**NOTE**

Exhibit B available from the lecturer by sending an email to jgady@ucsd.edu and requesting it.
### LIST OF ILLUSTRATIONS

<table>
<thead>
<tr>
<th>FIGURE</th>
<th>TITLE</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Overall Process</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>Preparatory Steps</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>Proposal Team Work</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>Program Work</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>System and Hardware Specification Template</td>
<td>8</td>
</tr>
<tr>
<td>6</td>
<td>Overview of the Traditional Structured Analysis Process</td>
<td>10</td>
</tr>
<tr>
<td>7</td>
<td>Ultimate System Diagram</td>
<td>11</td>
</tr>
<tr>
<td>8</td>
<td>System Environment</td>
<td>12</td>
</tr>
<tr>
<td>9</td>
<td>System Context Diagram</td>
<td>12</td>
</tr>
<tr>
<td>10</td>
<td>Function Sequence</td>
<td>13</td>
</tr>
<tr>
<td>11</td>
<td>Function Decomposition</td>
<td>14</td>
</tr>
<tr>
<td>12</td>
<td>System Life Cycle</td>
<td>14</td>
</tr>
<tr>
<td>13</td>
<td>Traditional Requirements Analysis Sheet</td>
<td>15</td>
</tr>
<tr>
<td>14</td>
<td>Product Entity Structure</td>
<td>16</td>
</tr>
<tr>
<td>15</td>
<td>Juxtaposition of RAS and N-square Diagrams</td>
<td>18</td>
</tr>
<tr>
<td>16</td>
<td>A Geometric View of the RAS Complete</td>
<td>23</td>
</tr>
<tr>
<td>17</td>
<td>RAS-Complete in Tabular Form</td>
<td>25</td>
</tr>
<tr>
<td>18</td>
<td>The System Relationship</td>
<td>26</td>
</tr>
<tr>
<td>19</td>
<td>SDD Structure</td>
<td>27</td>
</tr>
<tr>
<td>20</td>
<td>Context Diagram</td>
<td>30</td>
</tr>
<tr>
<td>21</td>
<td>Unified Modeling Language Overview</td>
<td>31</td>
</tr>
<tr>
<td>22</td>
<td>Hierarchical Relationship Between UML Dynamic Modeling Artifacts</td>
<td>33</td>
</tr>
<tr>
<td>23</td>
<td>Sequence Diagram Example</td>
<td>36</td>
</tr>
<tr>
<td>24</td>
<td>Communication Diagram Example</td>
<td>37</td>
</tr>
<tr>
<td>25</td>
<td>Activity Diagram Example</td>
<td>38</td>
</tr>
<tr>
<td>26</td>
<td>State Diagram Example</td>
<td>39</td>
</tr>
<tr>
<td>27</td>
<td>The UML Classifiers</td>
<td>40</td>
</tr>
<tr>
<td>28</td>
<td>Structural Relationships</td>
<td>41</td>
</tr>
<tr>
<td>29</td>
<td>Association Adornments</td>
<td>41</td>
</tr>
<tr>
<td>30</td>
<td>Software requirements specification template</td>
<td>43</td>
</tr>
<tr>
<td>31</td>
<td>Evolving Product Entity Structure</td>
<td>45</td>
</tr>
<tr>
<td>32</td>
<td>DoDAF Development Sequence</td>
<td>46</td>
</tr>
<tr>
<td>33</td>
<td>Requirements Traceability Across the Gap</td>
<td>49</td>
</tr>
<tr>
<td>34</td>
<td>Modeling Over the Years</td>
<td>50</td>
</tr>
<tr>
<td>35</td>
<td>The Approaching Merge</td>
<td>51</td>
</tr>
<tr>
<td>36</td>
<td>Tools Environment</td>
<td>53</td>
</tr>
<tr>
<td>37</td>
<td>Risk Matrix</td>
<td>56</td>
</tr>
<tr>
<td>38</td>
<td>Program Risk Tracking Chart</td>
<td>56</td>
</tr>
<tr>
<td>39</td>
<td>Verification Traceability</td>
<td>57</td>
</tr>
<tr>
<td>40</td>
<td>Specification Review and Approval</td>
<td>59</td>
</tr>
<tr>
<td>TABLE</td>
<td>TITLE</td>
<td>PAGE</td>
</tr>
<tr>
<td>-------</td>
<td>-------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>1</td>
<td>Independent and Combined SDD Appendices</td>
<td>44</td>
</tr>
<tr>
<td>2</td>
<td>Risk Probability of Occurrence Criteria</td>
<td>56</td>
</tr>
<tr>
<td>3</td>
<td>Risk Effects Criteria</td>
<td>56</td>
</tr>
<tr>
<td>4</td>
<td>Program Risk List</td>
<td>57</td>
</tr>
</tbody>
</table>
An Effective Specification Development Algorithm

The first order of business on any new program is the identification of requirements that collectively define the problem to be solved. This work can be completed in a timely and affordable way producing a quality definition of the problem space in terms of the minimum collection of requirements each one of which defines an essential characteristic of the system, or item thereof, to be developed. The target we should shoot for is all of the essential characteristics identified (no missed essential characteristics) and no unessential characteristics identified. Success in this process can be encouraged by the enterprise developing a set of specification templates and a corresponding set of data item descriptions coordinated with models selected to accomplish the requirements work and an effective suite of tools within which to capture the results. Every requirement that appears in every specification should be traceable to a model artifact from which it was derived. Before embarking on a new program, the enterprise needs to have selected this preferred set of models and tools and trained its staff to employ those models effectively entering the results in the tools suite. Every specification released is printed from the tools suite and should pass through a formal review and approval process as should any subsequent changes to any specification. As the models create work products, commonly simple diagrams, they should be saved in an organized fashion in paper or computer medias and formally released so as to be available for future system phases and modification projects.

Figure 1 offers a view of the overall process within which the preparatory work and definition activity that is discussed in the first section falls. Before entering a program, the enterprise should have prepared itself for the requirements work that will have to be done. The enterprise needs models to apply for the cases where the product is going to be implemented in computer software and in hardware. Both cases are covered in this tutorial followed by a discussion of integrated modeling.

Figure 1  Overall Process
1. Specification Templates and DIDs

Many system development organizations experience some difficulty in clearly identifying appropriate requirements for inclusion in specifications they must develop and they find it difficult to accomplish the work in an affordable and timely fashion. Over a period of the last two years the author developed an algorithm for improving system development organization ability in this work using templates and specially developed data item descriptions (DIDs). It requires some work to prepare the functional organization to support programs and some work on the part of proposal teams to accomplish initial analyses and extend the templates made available from the functional system engineering department to provide program-ready data, and work by the program teams starting with contract award and running through the period of time while specifications are being developed.

The goal of this specification algorithm is to provide for affordable and timely enterprise and program definition and documentation of new product technical requirements, the management and maintenance of related data, and publication and subsequent configuration control of the resulting documents.

The work required to implement the algorithm can be described in three preparatory steps: (1) enterprise engineering work, (2) system engineering generic work, and (3) proposal team work. The first two steps are illustrated in Figure 2. The numbers in the blocks coordinate with the steps in the algorithm. Recommended functional department responsibility for accomplishing the indicated tasks is noted at the lower left corner of the task blocks.

![Figure 2 Preparatory Steps](image)

1.1 Enterprise Engineering Work

Identify and staff an enterprise integration team (EIT) that is responsible for engineering the enterprise common process and acting as the process integration and optimization agent during its development and implementation on programs. The EIT should report to the enterprise executive.
1.1.1 The enterprise, through the efforts of the EIT, must develop a common process diagram that generically identifies all program work at a level of indenture that is adequate for making clear what work must be commonly done and allocating the corresponding work responsibilities to functional departments responsible for supplying programs with the necessary resources to accomplish that work well.

1.1.2 Allocate all work on the common process diagram to functional departments forming a task allocation matrix. This matrix establishes the requirements work that functional departments must be prepared to do on programs and any training that the functional departments are funded for and capable of performing must be focused on these tasks. This matrix covers the whole enterprise capability but in this section we are focused on doing the requirements work.

1.1.3 Functional departments collect all work allocated to them and build department manuals that explain what work must be done and provide links or descriptions for how to perform this work. One of the departments will be system engineering that will have responsibility for specification development and management on programs. For each task a functional department has responsibility, that department must identify work products that will result as a function of having accomplished the work on a program. EIT must integrate and optimize the evolving functional department work descriptions and work product identifications to ensure overall efficiency and effectiveness. All work products must have at least one user. Work products must be linked to the common process diagram tasks. Specifications are an example of a task work product and the work product of interest in this algorithm.

1.2 System Engineering Generic Work

1.2.1 To the extent that work products are documents, the responsible functional department must prepare a template containing the basic structure of the document in terms of generic paragraphing structure and calls for graphical images. In preparing for implementation of specification development and management work on programs in general, the functional system engineering department will select the specification standards that will be applied respecting the common customer base of the enterprise. They will review these standards and associated data item descriptions (DID), ensure that the system engineering department manual adequately covers specification standards the enterprise has chosen to respect, and build a set of specification templates (paragraph numbering and titles only), one for each kind of specification that will commonly have to be prepared on programs.

1.2.2 For each specification template defined, the system engineering department will determine one or more preferred modeling approaches for each kind of requirement in the template. The modeling approaches encouraged are the following relative to the primary kinds of specifications that will have to be developed:

<table>
<thead>
<tr>
<th>System Specification</th>
<th>Traditional Structured Analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hardware Performance Specification</td>
<td>Traditional Structured Analysis</td>
</tr>
<tr>
<td>Software Performance Specification, General</td>
<td>Unified Modeling Language (UML)</td>
</tr>
<tr>
<td>Software Performance Specification, Database</td>
<td>IDEF-1X</td>
</tr>
<tr>
<td>Software Performance Specification, DoD IS</td>
<td>DoDAF</td>
</tr>
</tbody>
</table>
1.2.3 Map models to templates such that for any of the specification s listed above there is a model identified for each paragraph of each specification. Ideally, all of the paragraphs of a particular kind of specification would employ models from the same family as suggested in the pairing above.

1.2.4 Map specialty engineering disciplines to specification templates in preparation for program mapping of these disciplines to specific product entities as a means of directing specialty engineering requirements analysis work.

1.2.5 For each template and model combination, prepare a DID communicating how the analyst will prepare the specification using a particular modeling approach and template. The DID must clearly show the connection between the modeling artifacts and the template paragraphing structure. The DID paragraphing structure must coordinate with the modeling components that are intended to yield derived requirements.

1.2.6 Map the functional departments that will be responsible for performing requirements analysis on programs to the template paragraphing structure for each kind of specification telling where the programs will acquire the analysts to accomplish the specification development work.

1.2.7 Prepare a template and DID for a system definition document (SDD) within which program structured analysis work products will be captured and configuration managed. These work products are to be captured in appendices of the document. An alternative is to capture the modeling artifacts within a computer tool that can be configuration managed.

1.2.8 Map the appendices of the SDD to the paragraphing structure of the templates telling in what appendix the corresponding work products will be captured.

1.2.9 Combine the functional department map (1.2.5), models map (1.2.3), and SDD appendix map (1.2.7) on a single matrix for each template and make these matrices available for program use.

1.2.10 EIT and the functional system engineering department must cooperate on selection of one or more computer tools or paper and pencil algorithms with which to accomplish requirements analysis on programs. Built a generic schema coordinated with preferred methods and models.

1.3 Proposal Work That Prepares the Program for Initiation

Proposal team work is illustrated in Figure 3. Blocks that do not have numbers coordinating them with the steps in this algorithm are not covered by the algorithm because they are not directly related to requirements analysis and specification development but these blocks add valuable context.
1.3.1 When beginning the proposal or program work, the manager should establish a program integration team (PIT) staffed by engineering, manufacturing, verification, logistics, and quality and a program business team (PBT) staffed by finance, contracts, scheduling, business information systems, and administration. Both of these teams should report to the program manager. These two teams could be combined as a staff function to the proposal manager but they will have integrating and optimization roles to play across the product oriented teams.

1.3.2 The PIT will perform an initial system analysis that will result in a clear understanding of any requirements provided by the customer, formatting of those requirements into alignment with the enterprise DID for a system specification, and adding to those requirements the results of their own system analysis work. A system environmental requirements analysis activity will expose a set of tailored standards covering the natural environment corresponding with spaces within which the system shall have to operate. A threat analysis will lead to exposure of hostile requirements. An interface analysis will identify external and top level internal interfaces that will be characterized in requirements statements. The result will be a preliminary system specification for submission with a proposal. If possible, this analysis work will be continued to develop all of the immediately subordinate specifications each of which will be the development responsibility of one of the top level integrated product and process teams (IPPT) to be identified and staffed subsequent to contract award.

1.3.3 The modeling work described in paragraph 1.3.2 will yield modeling artifacts from which requirements may be derived. The preliminary system specification development and any other specifications developed during the proposal effort will follow the pattern described in the
next paragraph. The second tier specification development may be delayed until a contract award but should precede the formation of the top tier IPPT. In all cases, requirements are derived from a model.

1.3.4 Requirements flowing from the structured analysis work will flow into a requirements analysis sheet (RAS) implemented in a computer database tool. All requirements entered into the RAS must include a traceability reference to the model from which they were derived.

1.3.5 Out of the initial PIT system analysis work will also come the preferred product entity breakdown diagram upon which the PIT, working in concert with integrated business team personnel, will construct overlays for organization responsibility breakdown (IPPT assignments), specification tree breakdown, engineering drawing breakdown, work breakdown (WBS), and manufacturing breakdown. The work breakdown will be handed off to the business team that will use it as the basis for building the program work definition, cost estimate, and IPPT work budgets. The IPPT will be assigned so as to align perfectly with the WBS making it possible to present to each IPPT leader as the teams are formed, a copy of the top level specification for which the team is responsible and the related budget and planning package for the whole WBS the team is responsible for as well as their top level schedule responsibilities encouraging the result that the IPPT leader may be held accountable for managing all aspects of the development of the entity assigned to the team.

1.3.6 The PIT will select a set of templates that correspond with the kinds of specifications that will have to be prepared on the program and the related DIDs that are coordinated with the models that will be applied in the analytical work. The PIT must also map specialty engineering disciplines to product entities to aid teams being formed to staff appropriately.

1.3.7 The PIT must take action to cause adequate computer tool seats to be allocated to the proposal team and accomplish any planning necessary for the subsequent program relative to the use of any requirements database tools and make any needed adjustments in database schema for the program.

1.3.8 The PIT shall capture the results of structured analysis work performed during the proposal activity in a preliminary system definition document (SDD) that will be used as the basis for subsequent lower tier analyses.

1.3.9 Any specifications developed in the proposal effort must be formally reviewed and approved by the proposal manager.

1.4 Work Subsequent to Contract Award

Specification related work to be accomplished subsequent to contract award is illustrated in Figure 4. This work is repetitious in nature progressively working down through the expanding architecture. Top-level teams may shred out during program work yielding sub-teams but in all cases, the top-level teams are responsible as managers for all lower tier team activity. This telescoping management responsibility is applicable throughout the team structures. During program performance, lower tier requirements analysis responsibility may be passed down
through the team structure with immediately superior team reviewing and approving of all immediately lower tier team specifications or the responsibility may be retained by the higher-level team but these decisions must be coordinated with the team budgets and staffing considerations arrived at during proposal work.

Figure 4 Program Work

1.5 The Preferred Templates

Ideally, the development organization would build a set of templates (paragraphing structure with no content) and data item descriptions (DID) that tell how to build a specification following the related template using a particular set of models. These should be maintained by the functional department in system engineering that has overall requirements and specification work responsibility. These should be available for reference or download by any new program.

Figure 5 offers a view of the preferred template for a system or hardware specification using traditional structured analysis as the modeling choice. The template is annotated with the preferred modeling artifact that will be used to identify the corresponding requirements and the functional department from which the program will obtain personnel to accomplish the related modeling work using the Figure 3 organizational structure. The data item description (DID) acronym in the model column means that the content is driven by the content of the DID used as the basis for the specification. The department references are cut at a very high level in this case and should be identified at one or two layers below this level but Figure 3 goes no deeper. The APP column gives the Appendix in the System Definition Document where the work products can be found.

The structure in Figure 5 can also be used for computer software requirements specifications (SRS) with paragraph 3.1 rewritten for UML and the model column updated to reflect UML artifacts.
A similar mapping should be provided for the software requirements specification (SRS) based on, in the author's view, the use of UML. Customers often require conformance to a DID supplied by them but may permit tailoring. The outline included in Figure 5 is a significantly tailored version of MIL-STD-961E to group all requirements so as to correspond with the modeling components contained in traditional structured analysis described in paragraph 2.2.1. k of the standard. The author believes this format will work with software as well but a different DID is required coordinated with the modeling approach selected (UML encouraged).
1.6 Modeling Work Product Capture Document

Programs should also be provided with a template for a System Definition Document (by whatever name) within which they can easily capture the results of all modeling work so that it can be preserved beyond the period of time when that work is actively being pursued. A later section describes this document. The appendices of the SDD are referred to in Figure 5 in the APP column and explained in Figure 20 and related text.

2 Structured Analysis

There are many models that can be used to accomplish an organized requirements identification effort that is preferred to an ad hoc method because it will most often hit the target noted earlier - identification of all of the essential characteristics and identification of no unessential characteristics. This process description encourages the use of traditional structured analysis in the near term to develop and identify the requirements for systems, hardware, facilities, real property improvements, and personnel actions captured in procedures as discussed in paragraph 2.1. One of the most difficult tasks in system development revolves around the relations between the system entities, the interfaces. This discussion of traditional structured analysis also contains a complete algorithm for identifying all system relations. Where the product is going to be implemented in computer software, unified modeling language (UML) is encouraged within a process context offered in paragraph 2.2. It is intended that a development organization should develop a process transformation roadmap needed to transition from this mixed method for the modeling work to a universal modeling approach, still evolving, that can be applied to all requirements work at the earliest possible. See paragraph 2.3 for a discussion of integrated modeling.

2.1 Traditional Structured Analysis

Figure 6 illustrates an overview of the traditional structured analysis process (TSA). The eleven numbered steps are briefly introduced followed by additional details in subordinate paragraphs.

1. Understand the User Requirements - Through conversation with the user and/or reading a user requirements statement or specification, the developer tries to understand the user's need. This is not ever easy because the user is able to explain only what their mission interests are and the developer needs hard engineering data.
2. Decompose - Users commonly have problems that are too grand to be easily understood in a single small document or simple diagram. These problems commonly have to be decomposed or partitioned into a collection of smaller related problems.
3. Functional Flow Diagram - The TSA approach employs some form of functional analysis as the decomposition medium.
4. Performance Requirements Analysis - The functions are translated into performance requirements.
5. Requirements Analysis Sheet (RAS) - The strings of functions, performance requirements, and product entities to which they are allocated appear as rows in the RAS. The RAS also is used to capture all system requirements linked to the model from which they were derived.
6. Requirements Allocation - Performance requirements are allocated to product entities in the RAS.
7. Product Entity Structure - The physical and logical entities that comprise the system are arranged in a hierarchical structure that is used as the basis for WBS, specification identification, team assignments, and many other program applications.
8. N-Square Diagram - The interfaces that must exist between the product entities are identified through a pair-wise analysis of all possible interface relationships.
9. Environmental requirements for the expanding product entities are determined through application of a three-layer model.
10. Specialty engineering requirements are identified by a group of specialist linked to the product entity structure by a specialty engineering matrix.
11. This process is applied iteratively as lower tier entities are identified through functional analysis.

Figure 6  Overview of the Traditional Structured Analysis Process
2.1.1 A System Defined

A man-made system is a collection of entities that are meant to interact in predictable ways with an environment and with each other via relations between them to achieve a useful function identified and articulated by a customer as a mission need statement. Therefore, systems are composed of entities and relations between the entities. The system is intended to satisfy the mission need statement, the system’s ultimate function, depicted on system diagrams as a rectangular block titled System Need and identified with a functional identifier F. The need is allocated to the system depicted on system entity model by a rectangular block named “system” (or a particular system) and identified with a product entity identifier A.

A system interacts within an environment as shown in Figure 7. The environment for every system is everything in the Universe (U) less the entities that are part of the system product entity structure (Q = U - A). One can reduce the scope of the environment to those elements that will have some influence on the system. The line joining the system and environment in Figure 7 (I2) indicates the relations between the two (external interfaces). The line joining the system on both terminals (I1) indicates the internal relations between system entities (internal interfaces) yet to be defined within the system.

![Figure 7 Ultimate System Diagram](image)

2.1.2 The System Environment

The environment for any system is composed of the subsets illustrated in Figure 8. While all environmental effects on the system are relations, they may be partitioned between those that are commonly considered environmental stresses and the cooperative environmental elements that are treated as external interfaces commonly developed by a pair of teams or contractors responsible for the terminal product entities.

A context diagram, such as that shown in Figure 9, even though similar in nature to Figure 7, offers a useful simple model for focusing attention on identifying all external relations. Some of these terminators will be natural, non-cooperative, or induced environmental stresses. Others include hostile stresses determined through a system threat analysis as well as both stresses and useful relations with cooperative systems. Though this diagram was conceived as the beginning of the modern structured analysis model, it has a useful purpose in TSA as a means of viewing all of the inside-outside relationships and in UML as an organizer of use cases. It can be used to make a first impression in the line I2 in Figure 7.
The system natural environment is determined by defining all of the spaces within which the system will be employed based on an analysis of the intended mission and basing concept. The spaces are coordinated with a set of environmental standards. Each standard is studied for necessary content and the remainder tailored out. Each selected parameter is then studied for an appropriate range. The system natural environment is then the union of the selected parameters from the selected standards.

The non-cooperative environment is defined by determining what stresses will be applied to the system from man made systems which are neither hostile nor cooperative. An example of non-cooperative stress is electromagnetic energy. Self-induced environmental stresses are not easily
determined at the system level because one needs to understand energy sources and other stressors within the system determined as part of the design of end items.

System cooperative environmental relations are defined by determining how the system to be developed will associate with other friendly systems already in existence or being developed. These associations may be coupled into or out of the system in terms of information, physical association, materials, or energy.

2.1.3 System Functionality

A function is a necessary activity for a system to perform. It may be static, dynamic, or both. It should be named using an action verb followed by a noun. A function is depicted in modeling the system as a rectangular block identified by an action verb name centered in the box and a function identifier (ID) in the lower right corner. The ultimate function for any system is the customer need the name of which is the need statement possibly paraphrased to fit into the space provided coupled with a function identifier F.

Two or more functions can be linked together using directed line segments to show a sequence of functions. In Figure 10 the understanding is that function F1 must be accomplished before function F2. Combinatorial symbols may be added to permit more complex sequential relationships. The combinatorial symbols encouraged are AND, inclusive OR (IOR), and exclusive OR (XOR) with the common logical meanings. Enhanced functional flow block diagramming adds loop (repeat a function until a specific outcome has been attained) and iterate (repeat a function a specific number of times) combinatorial symbols that can be useful. Diagrams so constructed are called functional flow diagrams. These diagrams may be oriented on the page with their primary flow axis arranged horizontally or vertically with the flow in either direction.

![Figure 10 Function Sequence](image)

For any function with identifier F@ (where @ is a string of length n (including n=0) composed of characters from the set {A through Z less O}U(a through z less I)U(0 through 9)}) there may exist one or more subordinate functions F@# (where # is a single character from the same set identified above which differentiates other functions at that level from one another). This is illustrated in Figure 11. Every function need not have an expansion. There is no need to assign function identifiers in alpha numeric sequence on a page of the diagram but it helps the human to use the diagram if they are assigned initially coordinated as much as possible with the function sequence. If a function is deleted subsequent to a release of the diagram, that identifier should not be used again. If the number of functions on any one diagram exceeds the maximum number of symbols available, 60, change the diagram to reduce the number to less than 60.
Ideally, whoever accomplishes the initial analysis of the need, would do so using the functional analysis process described here where the first decomposition is the system life cycle as shown in Figure 12 and the second is an expansion of the life cycle function “Use System” (F47) to expose the top level operational intent and initial content of the user requirements documentation or preliminary system specification. If the customer or other initial analysis agent applies an unstructured or ad hoc approach, then the development organization may have to accomplish a functional analysis and try to map the requirements identified by the customer (user and acquisition agent) to the functionality exposed when they do accomplish this work.
Ideally, the development organization would extend the functional analysis into the remainder of the system life cycle functions as well as the Use System function determining appropriate resources for the process steps of the development program and the product system being developed such that the physical product delivered will be jointly optimized relative to its product and process. Commonly, process functions do, or should, influence product functions and the corresponding product entities needed as well as the opposite case.

2.1.4 Performance Requirements Derivation and Allocation

The functions identified in the functional flow diagram must be translated into performance requirements that tell what the system and its parts must do and how well it must do them as shown in Figure 13. These statements can be first developed as primitive statements for example phrased as “Velocity ≥ 600 knots” without complete sentence structures and subsequently transformed into complete sentences in the chosen language. Traditionally, a requirements analysis sheet (RAS) has been used to capture the function identification, the primitive performance requirements statements, and the allocation of these performance requirements to product entities. One could allocate the function names directly to architecture but often one finds a one-to-many allocation result this way whereas allocation of performance requirements tends to follow a one-to-one pattern. To fully characterize a function it may require identification of multiple performance requirements and these several performance requirements may be allocated to different product entities.

The RAS as traditionally used is incomplete, unfortunately, and this discussion will use a RAS complete. We will show how the three kinds of constraints can also be captured in the context of Figure 13 shortly. The intent is to be able to capture all requirements in a RAS linked to a modeling artifact implemented in a computer application.

<table>
<thead>
<tr>
<th>FUNCTION</th>
<th>PRODUCT ENTITY</th>
<th>REQUIREMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>MID</td>
<td>NAME</td>
<td>PID</td>
</tr>
<tr>
<td>F123</td>
<td>Provide Thrust</td>
<td>A12</td>
</tr>
</tbody>
</table>

Figure 13 Traditional Requirements Analysis Sheet

2.1.5 Product Entity Structure

System functionality is accomplished by physical entities that form part of the physical system. These entities in the system, in the aggregate, comprise the product entity structure that the author formerly referred to as the system architecture. The word architecture has taken on a significantly broader meaning in recent years convincing the author that what he formally referred to as architecture should now be referred to by the more isolating term product entity.
structure. The function $F$ is accomplished by product entity structure element A, the system, by definition as shown in Figure 7. Lower tier entities should be exposed following Sullivan’s encouragement of form follows function. Trade studies may be appropriate to make hard decisions on the best implementation of a particular function in early program phases. The physical entities that accomplish functionality can be partitioned into five classes: (1) hardware, (2) computer software, (3) facilities and improvements to real property, (4) procedural definition, and (5) humans following procedures or acting autonomously. We could merge the last two into one. The relationship between functions, the physical entities in the product entity structure, and the corresponding performance requirements is depicted on a simple requirements analysis sheet as shown in Figure 13.

The aggregate product entity structure for a system is illustrated in a hierarchical model connecting the product entities arranged into a breakdown block diagram as illustrated in Figure 14. The entity identifiers follow the same pattern as the function identifiers explained earlier beginning with a letter A because of the author's previous use of the word architecture for this view. The entities stream out of the RAS available for structuring into the product entity block diagram. Ideally, this work would be accomplished by a team of people representing hardware and software engineering, manufacturing, procurement, and verification with strong system/program leadership by the PIT. Initial functional analysis and allocation must concentrate on understanding user mission needs. This will generally require intense interaction with the user, ideally using system models to encourage mutual understanding. Alternative ways of implementing functionality with different product entities should be considered and where the decision is very difficult they should be subjected to a trade study.

![Figure 14 Product Entity Structure](image-url)
2.1.6 Allocation Pacing Alternatives

The conduct of the functional analysis and allocation work can be paced in one of four different ways:

(1) Instant – As soon as functions and/or their corresponding performance requirements are identified, they must be immediately allocated to the expanding product entity structure.

(2) Terminal - All of the functional analysis must be complete before any functions and/or their corresponding performance requirements can be allocated to product entities.

(3) Layered - The analyst completes one layer of the expansion of a functions and must allocate all of the exposed functionality and/or the corresponding performance requirements to product entities before further expanding that function. This works best if all product entities related to a layer are defined in terms of their requirements and design concept before pursuing the next functional layer and its allocation but there generally is not sufficient time available and one must accomplish a lot of this work in parallel leading to some risk and interaction. The layered approach has been popularized by Mr. Bernard Morais and the late Dr. Brian Mar under the title FRAT.

(4) Progressive – The analyst completes several layers of the functional analysis without allocating any of it to product entities. At a layer guided by experience the analyst begins allocating performance requirements derived from higher tier functionality to product entities. Design concepts are defined for the product entities at the higher levels and these act to both guide and constrain lower tier functionality progressively. Allocation is delayed throughout the analysis such that higher tier design concepts help to steer lower tier functional analysis tending to isolate iteration to one structure (functionality, architecture, or allocation) at a time. The two extreme cases (1 and 2) are flawed due to a need to iterate F, A, and allocations excessively in the case of 1 and a common need to significantly change lower tier functionality after the higher tier allocations begin in the case of 2. The progressive approaches either by layer or guided by experience will generally produce better results with less modeling hysteresis.

There exists a downward limit in decomposition or expansion of the functional portrayal. This limit for any one branch in the expansion is best determined on the product entity plane based on the analyst’s understanding of the problems related to the lowest tier product entities. Where all of the lowest tier entities are fully characterized supporting procurement or in-house design, the functional analysis work can be reduced to maintenance of the models and related data. There is one more layer of requirements related to parts, materials, and processes but that is driven by design decisions during synthesis and commonly has already been completed by those responsible for the sources of these entities. The logistics analysis process may require a switch to a process diagram and if the progressive allocation approach described has been followed the functional flow will have been migrated toward a process diagram at the lower tiers.
Figure 15 illustrates a geometric view of the process of deriving performance requirements from functions and the allocation of those performance requirements to product entities. For example, a performance requirement has been derived from F4713 and allocated to product entity A11. This plane represents the normal RAS used only for function allocations. As suggested by the other structures we will use the geometric structure to expand the RAS to cover all requirements.

![Diagram of RAS and N-square Diagrams]

**Figure 15** Juxtaposition of RAS and N-square Diagrams

### 2.1.7 System Relations

As the product entity structure is formed, the analyst can begin to identify the needed internal relations between the system entities by using an n-square diagram where the product entities are identified down the diagonal at some level of indenture. For a given analysis where the number of entities being studied for interface relations in an n-square diagram is N the largest possible number of relations is N(N-1) counting each direction as one possibility between each pair of entities. Interfaces between these entities is pre-determined by the way that functionality is allocated to the entities. Therefore, one may explore the list of function-product entity pairs
associated with each product entity in the n-square diagram. This is referred to as a pair-wise analysis of the n-square diagram intersections.

Figure 15 includes an n-square diagram with only half of the square showing (the remainder hiding behind the other structures in the diagram). The diagonal (containing the product entities we are interested in accomplishing the pair-wise analysis on) has been aligned with the product entity axis of the function-product entity matrix (which corresponds to the simple RAS of Figure 13).

The process for marking the intersections of the function driven relations matrix (n-square plane) entails a pair-wise analysis of the function-product entity pairing relationships marked on this matrix. Interface Ix is encouraged by the conclusion that if F4711 maps to A13 and F4712 maps to A11 then there is a possible demand for an interface between A11 and A13 to implement that relationship. So we map functions to product entities but we map F-A pairs to interfaces and those interfaces are pre-determined by the way we allocate functions to the product entities.

If the system is a modern war ship or the system that will take man to Mars, a pair-wise analysis of the function driven relations matrix would be beyond human comprehension if attempted all at one time, We can, however, partition the analysis work to one interface expansion at a time and it is not so overwhelming. At any one level of product entity granularity, we can explore one layer of product entity structure expansion for internal interfaces within the parent item. If there are five subordinate entities then the number of possible interfaces to be examined in a pair-wise fashion would be 5x4 or 20. Similarly, external interfaces can be analyzed individually. Of course, it will still be necessary to accomplish considerable interface integration work because of the partitioned process. This process will go very fast if the analyst is very familiar with the problem space and the evolving solution concepts. The complete algorithm is extended in subsequent paragraphs.

The requirements analysis sheet (RAS) identifies every possible pairing of functions and architecture entities. We may sort this listing so that all of the functions (or performance requirements derived from those functions) allocated to each entity are grouped by entity. Then we can pile up the allocations onto the product entity squares on the diagonal of a physical n-square diagram as suggested in Figure 15.

The discussion so far has dealt only with internal interface identification all defined on the function driven relations matrix of Figure 15. To cover external interfaces we add the larger n-square diagram on Figure 15. The diagonal in this case includes all of the product entities plus all of the external entities in the cooperative environment. We can identify relations between these external and internal entities in the same way we did the internal ones. The association of function F4712 to cooperative environment entity QC1 and allocation of a function to A12 may define a need for an interface I2y.
2.1.8 Environmental Relations Algorithm

2.1.8.1 System Environmental Relations

The system environment consists of all entities in the Universe less those that are in the system. That is, Q=(U-A) where Q is the environment, U is the Universe, and A is the product entity structure of the system being developed. It is convenient to partition all system level environmental relations into the sets illustrated in Figure 8. The system cooperative environment (QC) can actually be treated as an external interface and can be developed using the algorithm covered in Paragraph 2.1.7 very effectively. External cooperative systems are simply located on an extension of the product entity axis forming the cooperative environment axis. The hostile environment (QH) can best be understood through analysis of threats posed by hostile forces. The non-cooperative environment will yield to the same thought process applied in the threat analysis except that the stresses applied to the system are not applied for hostile purposes, rather simply because the system being developed is sharing a common operating space with other systems. Electromagnetic interference is an example of the stresses applied in this set.

System time (QN2) is studied using time lines oriented about the functions that the system must satisfy. When we allocate those function (or their corresponding performance requirements) to architectural entities the timing requirements corresponding to the functions are applied to the entity as timing requirements.

System space (QN1) is defined through mission analysis such that it is determined in what volumetric spaces of the Earth (surface, subsurface, and aerodynamic) and/or surrounding space and/or distant bodies the system shall function within, on, or around. For each space, that space is teemed up with one or more natural environmental standards that define that space. Each of these standards is then studied to determine which natural environmental (QN3) parameters included in the standard shall apply to the system being developed. Those that do not apply are tailored out of the standard. The selected parameters are then studied to ensure that the range of values is appropriate for the system. If the range in the standard is too broad it is tailored to narrow the range of values to that for which the system shall be designed. This process is repeated for each standard linked to a system operating space.

Figure 15 also extends the RAS concept to include environmental requirements analysis. The system environment as depicted in Figure 7 is illustrated at the diagram “origin” on the environmental axis that happens to coincide with the architecture “origin” that corresponds to the whole physical system A.

2.1.8.2 End Item Service Use Profile

An end item is a major element of a system that generally retains its physical configuration throughout its mission performance and has an end use function. The end items may remain fixed in place during use or move over great distances and maneuver within the system spaces as a function of the system mission and the end item’s application in the system. Each end item should be designed to endure only those natural environmental stresses anticipated so it is necessary to determine what subset of the system natural environment each end item shall be
exposed to. To accomplish this, one must create a physical process flow diagram. This is not the same as the functional flow diagram used to identify system architecture and performance requirements. The blocks on a functional flow diagram represent things that have to happen whereas the block of a process diagram represent a real world analogy. You cannot profitably consider system entities flowing through the functional flow diagram, indeed we are using the diagram to determine what those entities should be. However, we can imagine the system product entities flowing in the process diagram where each process acts as a transformation on the system entities.

The first step in defining end item environmental relations is to map the system environmental parameters to the process steps at some level of indenture, generally at a level where the environmental map does not change significantly below that process level. This work defines an environment for each process. The next step is to map the architectural entities to the process blocks. If an entity only maps to a single process step, it simply inherits the process environmental set. If, as is so often the case, the architectural entity maps to two or more process blocks, then it will be necessary to apply some kind of integrating process to any differences in environmental stresses and their values observed between the two or more process blocks. The rule most often selected initially is to pick the worst-case range for each parameter across the process values being evaluated. If this rule does not adversely influence system cost, then it is an adequate solution. If this approach either results in an adversely narrowed system solution space, then an alternative to “worst case” must be derived. Often time lines will show that the problem environmental stress will be applied over such a short time as to be insignificant. In other cases, one may find that the problem can be narrowed to some particular combinations of values that are very unlikely to occur. If the problem is intractable, it may be necessary to restrict one or more system environmental parameters more severely than is currently being done. Generally, this will have an adverse effect on system performance.

The self induced environment (QI) can best be studied and defined at the end item level since it is end items which contain the sources of energy and other stresses of interest which will reflect commonly through a natural environmental parameter right back into the system. Since the self-induced stresses are commonly greater in magnitude than the corresponding natural stresses for that same parameter, these induced relation values have the effect of extending the range defined through the application of the end item service use profile algorithm discussed above.

2.1.8.3 Component Environmental Relations

The environmental relations appropriate for components installed within end items are simply the end item environmental stresses if the end item has no altering effect on those stresses and all spaces within the end item offer the same environmental stresses to components installed within them. Where an end item does have a modifying effect on end item stresses but all spaces within and upon the end item offer the same stresses, it is necessary to determine the end item design effect on end item environmental stresses and the result is the set of installed component environmental stresses. The most complex case occurs when the end item must be partitioned into two or more spaces more often called zones of common environmental stresses. The value of each end item parameter must be determined for each zone thus defining the environment for each zone. Then one maps the components to the zones and they inherit the zone environments.
Commonly, the job is not complete at this point because it is found that there is no zone within which one or more components can be installed in a particular end item that will cause their environmental stress limits to be satisfied. When this happens, it is necessary to either change the component environmental specification values or include an environmental control system as an added entity into which the components with the environmental range shortfall problem are placed.

2.1.8.4 Environmental Impact

The environmental effects discussed in the three previous cases deal with environmental stresses the environment will apply to the system but there may be cases of the opposite direction that will cause damage to the environment. Once identified by someone skilled in environmental impact, these can be treated like safety hazards to be mitigated through re-design, procedural controls, or compensating environmental actions. In the case of military systems it is very difficult to mitigate the damaging effects of warfare but these systems can also have damaging effects on the environment in peaceful use such as training and maintenance. Often these materials are just naturally dangerous to be around as illustrated in the difficulties observed in the base closure efforts where many adverse environmental effects have had to be identified and mitigated.

2.1.9 Specialty Engineering and RAS Complete

The system engineering agent for the system must build a list of all of the specialty engineering disciplines that will be applied in the development of the system. A specialty engineering scoping matrix should be prepared between specialty engineering disciplines that may be required in development of the system and the product entities. This will help to determine team staffing needs in that area and connect people in those disciplines with a need to do specialty engineering requirements analysis for the indicated items. Figure 16 adds one more plane, the specialty engineering scoping matrix, to the construct previously illustrated in Figure 15 providing for allocation of specialty engineering disciplines listed to architecture.

Specialty discipline H7 is shown mapped to architecture item A11. This must be followed by analyst definition of one or more discipline H7 requirements that will flow into the specification corresponding to the product entity. The structure exposed in Figure 16 is a complete RAS showing all of the important requirements related relationships supporting the requirements analysis process leading to the identification of every kind of requirement appropriate to the system and hardware specifications and all of them linked to and derived from a model.
The fact that an aircraft airframe will have to be checked for alignment during manufacture and after hard use (hard landing or pulling excessive g’s in flight, for example) identifies a need for a relation between the airframe and the equipment which will be used to accomplish the alignment. Today this will generally call for some form of laser optical application so the airframe would have to either include targets, detectors, or mirrors or provisions for these to be applied. The manufacturing and maintenance engineers would have to consider all of the ways there may have to be relations between support equipment and the operational entity. There may be other cases where these disciplines have to call for internal relations within the system entities. For example, if the system must include built in test (BIT), there will have to be relations between most of the on board equipment and some entity that concentrates the BIT effects for display to operations and/or maintenance personnel.
The needs of operations personnel, such as aircraft pilots, locomotive engineers, ship captains, and automobile drivers provide a tremendously rich class of entity relation possibilities. The physical relations are always fairly simple in that the human operator has only his/her senses, mental acuity, and physical strength through which to interact with the system. This problem is made much more complex, however, because not all humans will react in precisely the same way to a particular stimulus. It will be necessary to determine the complete data set that the human will require under all operating conditions and in what way the human shall influence system behavior in terms of controls. Operator sequence diagrams, built like a UML activity diagram with swim lanes, can be useful in doing this work.

Every one of the specialty engineering disciplines selected for the program must be evaluated for entity relation driving potential and those persons doing that work alerted to their responsibilities in identifying relations for further consideration by the whole team.

2.1.10 RAS-Complete in Table Form

The results from the analyses noted in prior text must be captured on its way into program specifications. Certainly, the most advantageous way to do that is in a computer database systems such as DOORS, CORE, or SLATE. Therefore we would expect that some form of tabular structure would suffice. Figure 17 is offered as the candidate view of this table for use during development in capturing the relationship between model, requirements, product, and document entities. The model ID (MID) identifies the model from which the requirement was derived. The requirement columns identify the requirement ID (RID) assigned by the computer system for use in traceability. The RID is a made-up computer-assigned unique field using a base 60 numbering system in this example but a hexa-decimal implementation is probably more common. Ideally the requirement statement should be captured in primitive form (attribute, relation, value, and units) wherever possible with different fields for each component of the string. The primitive form is shown concatenated in Figure 17. The final column pair offers specification paragraph number and title.

The sample data included is ordered by model ID alphanumerically separating the data into the four kinds of requirements found in a system or hardware specification. The lists the MIDs respected by the author is still in a state of change as different RAS-Complete possibilities are explored. This may explain the apparent unthinking selection of H and Q for specialty engineering and environmental requirements, respectively. The intent is to identify all possible modeling artifacts with a letter as a source from which every conceivable requirement may be derived. Model letters for UML artifacts have been included in the set and will be introduced later.

This view provides clear traceability between the models from which the requirements were derived and the product entities to which they were allocated for all of the requirements, not just the performance requirements. What the author calls lateral traceability is captured in the database implementing the RAS-Complete. It is also a simple matter to link the rows in the matrix in a database to the corresponding verification requirements as well as the tasks to which they are allocated and their corresponding plans, procedures, and reports. Vertical traceability is, of course, simply a matter of relating the unique RIDs from pairs of requirements in specification parent-child relationships identified by their PID.
### Table 1: RAS-Complete in Tabular Form

<table>
<thead>
<tr>
<th>MODEL ENTITY</th>
<th>REQUIREMENT ENTITY</th>
<th>PRODUCT ENTITY</th>
<th>DOCUMENT ENTITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>H</td>
<td>Specialty Engineering Disciplines</td>
<td>A Product System</td>
<td></td>
</tr>
<tr>
<td>H11</td>
<td>Reliability</td>
<td>A Sensor Subsystem</td>
<td>3.1.5 Reliability</td>
</tr>
<tr>
<td>H12</td>
<td>Maintainability</td>
<td>A Sensor Subsystem</td>
<td>3.1.6 Maintainability</td>
</tr>
<tr>
<td>I</td>
<td>System Interface</td>
<td>A Product System</td>
<td></td>
</tr>
<tr>
<td>I1</td>
<td>Internal Interface</td>
<td>A Product System</td>
<td></td>
</tr>
<tr>
<td>I181</td>
<td>Aggregate Signal Feed Source Impedance</td>
<td>A Sensor Subsystem</td>
<td></td>
</tr>
<tr>
<td>I182</td>
<td>Aggregate Signal Feed Load Impedance</td>
<td>A Analysis and Reporting Subsystem</td>
<td></td>
</tr>
<tr>
<td>Q</td>
<td>System Environment</td>
<td>A Product System</td>
<td></td>
</tr>
<tr>
<td>QH</td>
<td>Hostile Environment</td>
<td>A Product System</td>
<td></td>
</tr>
<tr>
<td>QI</td>
<td>Self-Induced Environmental Stresses</td>
<td>A Product System</td>
<td></td>
</tr>
<tr>
<td>QN</td>
<td>Natural Environment</td>
<td>A Product System</td>
<td></td>
</tr>
<tr>
<td>QN1</td>
<td>Temperature</td>
<td>A Product System</td>
<td></td>
</tr>
<tr>
<td>QX</td>
<td>Non-Cooperative Environmental Stresses</td>
<td>A Product System</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 17** RAS-Complete in Tabular Form

#### 2.1.11 Traditional Structured Analysis Summary

In summary, a system is defined by identifying its functionality starting with the need (F), allocating that functionality to entities that become part of the system architecture (A). These entities that form the system architecture are selected by determining the performance requirements that the system must satisfy to meet the top-level customer’s need. The pairs of functions and product entity allocations pre-determine how the entities will have to relate to each other through interfaces (I) between the product entities. The environmental elements (E) are defined at the system level in terms of the spaces within which the system must function and the corresponding characteristics of those spaces drawn from appropriate environmental standards covering those spaces. As depicted in Figure 18, the traditional structured analysis effort attempts to define the most cost effective solution such that in N cycles of the process axis of the physical system (generally cyclical in the interest of reuse of system elements) the relation P (process) maps the cross product of the power sets of architecture (A*), interface (I*), and environment (Q*) to the function set F such that F is covered.
For every process Pi, there exists a combination of architecture, interface relations, and environmental stresses such that some subset of the function set is covered or accomplished. The power sets of these entities include all of the possible subsets of these entities within their own set thus the power set of A includes every useful combination of product entities relative to every process step. Useless subsets are also included in the power set as well, of course. It is important that the functions be covered in the correct order determined by the sequence of the processes linked together in the process axis. If all of the functions are satisfied in N revolutions of the process axis as planned, then we may say that the system is consistent relative to the use of its product entities, interfaces, and environmental stresses. If there are product entities that are not used in the process or some that are needed but not available we may not have the optimum product entity structure.

This whole process happens in practice somewhat backwards in that for an unprecedented system, one begins the development process only knowing the ultimate function, the need, and must expand everything from that one perspective.

2.1.12 SDD Content and Format

When used to support the application of traditional structured analysis on a program, the preferred SDD format consists of a main body and seven appendices, each providing a capture point for the work products of one of the several fundamental analytical system requirements analysis process areas. Figure 19 shows how the document is structured. A series of seven interactive system analysis activities feed the development of the appended data explained in subordinate paragraphs. The appended data then becomes the basis for lower tier analysis that produces content for the lower tier specifications and adds to the appended data.
2.1.12.1 Document Main Body

The main body simply contains a table of contents, list of illustrations, and list of tables for the document plus it should provide text explaining the capture of work products in the seven work areas during system and lower tier analyses. The body should also explain that the SDD couples the structured analysis work and its work products to specification content as guided by the selected specification standard templates.

2.1.12.2 Appendix A, Functional Analysis

This appendix captures the functional flow diagram starting with the identification of the system need and the life cycle flow diagram. The Use System Function is initially decomposed progressively to expose more details about the user need. For each block in the functional flow diagram, there should be one line in the function dictionary also contained in Appendix A.

2.1.12.3 Appendix B, System Environment Analysis

The environment consists of several subsets of stresses that are applied to the system. This appendix identifies and characterizes them. Timelines capture critical timing requirements. The spaces within which elements of the system must function are identified and the corresponding environmental stresses defined in terms of standards that describe those spaces. Service use profile analysis is applied to uncover end item environmental requirements. Finally, zoning of end items exposes component environmental requirements.
2.1.12.4 Appendix C, System Product Entity Analysis

The system product entities result from the allocation of functionality to things. As these pairs are defined on the function-product entity matrix, they must be entered into the product entity structure diagram. This work should be accomplished by a team of people knowledgeable in system, hardware, and software engineering, manufacturing engineering, verification engineering, logistics, material and procurement, and logistics in order to evolve an optimum product entity structure which will be universally respected on the program. This product entity structure is also the basis for the specification tree. Each item on the tree must have a responsible agent identified, a template selected, and a release date established. This structure should also be the basis for any IPPT established on the program. It is also the basis for the WBS so the SOW and IMP align perfectly with the teaming structure applied on the program.

2.1.12.5 Appendix D, System Interface Analysis

Interfaces are identified by pair-wise evaluation of function allocations to product entities using an n-square diagram. This appendix identifies all interface needs internal to the system as well as externally to the cooperative systems identified in Appendix B.

2.1.12.6 Appendix E, Specialty Engineering Definition Analysis

Appendix E provides a space in which system engineers can capture their work directed at identifying the specialty engineering disciplines that will have to accomplish work on the various entities in the system product entity structure to define the appropriate requirements and subsequently the needed analyses to confirm that those requirements are being satisfied. A specialty engineering scoping matrix is used to report the results of that analysis.

2.1.12.7 Appendix F, System Process Analysis

Appendix F captures the results of a physical process analysis in the form of a process flow diagram. This is used by logistics engineers to drive out requirements related to training, support equipment, maintenance procedures (tech data content), and spares consumption. It is also needed to complete the environmental use profile study reported upon in Appendix B that drives environmental requirements for end items.

2.1.12.8 Appendix G, Requirements Analysis Sheet

The exposed functions are listed in the Requirements Analysis Sheet (RAS) contained in this appendix. Related performance requirements are defined and allocated to a product entity. These performance requirements have to have a paragraph number assigned, title identified, and they can be outputted into a specification following a particular template. That part of the work can be done inside a requirements database system. Ideally, all of this work would take place within a requirements database tool but some organizations may find it preferable for their purposes to do the traditional structured analysis work using pencil and paper followed by capture of the resulting requirements in a word processor or a computer database tool from which specifications can be generated.
In keeping with the Integrated RAS idea advanced in Paragraph 2.1.10, the RAS is not restricted to performance requirements. Specialty engineering, interface, and environmental requirements can also be included so that every requirement appearing in every specification on a program will transition from the analytical model from which it was derived into a specification via the requirements analysis sheet.

2.1.13 Team Activity During Requirements Work

The PIT is responsible for accomplishing all requirements analysis work focused on developing the system specification and the immediately subordinate specifications that will be the top-level specifications for the top level IPPT. In general, this analysis work will be accomplished using traditional structured analysis following the pattern described in this section. PIT initiates the analysis capturing the work products in the SDD thus initiating that document. Requirements derived from the modeling work are entered into the database application initiating the requirements capture. Any customer requirements documentation is also entered into the database and traceability established between these requirements and those developed from modeling efforts that appear in the system specification. Traceability is continued down to the subordinate specifications to be handed over to the top-level teams when formed.

The PIT identifies all system external interfaces and defines them in the system specification or the beginning of an interface control document. Internal interfaces are also identified and defined for the first tier entities below the system level and the next lower tier in order to complete the internal interface definition for the first tier. All requirements are entered into the requirements database application and traceability entered.

The PIT develops a specialty engineering scoping matrix and maps the needed disciplines to the product entities identified and coordinates the indicated domain experts to derive requirements for the system and top-level end item specifications. The system level environmental requirements are derived from system spaces identified and mapped to corresponding tailored standards.

With the top-level specifications completed, the PIT can bring the top-level IPPT aboard. As those teams assemble and become familiar with their specification and the program planning data prepared by the proposal team, they continue the requirements analysis process relative to the functionality of their product entity. The primary role of the PIT switches to integration and optimization across the IPPT. The teams enter requirements data into the requirements database and maintain traceability. As this process continues, it may become apparent that lower tier teams are required in which case the parent team takes over system responsibilities for them. The parent team in all cases must develop the top-level specification for any new team. Also, at some point, a team will identify an allocation of functionality to computer software and the continued analysis of that entity should switch to UML.
2.2 UML

2.2.1 Entry Analysis and Overview

While it is encouraged that the enterprise apply TSA today as the entry modeling technique, it is entirely possible to initially enter the problem space analysis for a system that will be implemented primarily in software with UML rather than TSA or SysML. The suggested process starts at the top with the problem expressed in the system need and illustrated in a context diagram, borrowed from modern structured analysis, like that shown in Figure 20. The context diagram expresses relationships between the system, represented by the bubble, and terminators, representing outside entities deriving benefits from the system and supplying things needed by the system to function, generally information in a computer software system but more general in a system that will be implemented using a collection of technologies. Figure 20 is an alternative to the traditional depiction of the general system as a block labeled System interacting with a block named Environment.

![Diagram of Context Diagram](image)

Figure 20 Context Diagram

While UML could be the entry modeling approach at the system level following the approach discussed in this section, the discussion to follow is based on the assumption that the problem space will be entered using TSA with a recognition at some point of a need to switch to UML as entities are identified that must be developed in software. Figure 21 illustrates a process for applying UML starting at whatever level in the system the program chooses to start applying it. Generally, this will be some level below the system level based on allocation of higher tier functionality to a software entity.

In a given system, the initial analysis may have identified one or more entities that will be implemented in computer software so for each of these separate entities one should build a context diagram. It should be noted that the context diagram was popularized in modern structured analysis and was not adopted by UML but it is a useful artifact with which to identify the inside-outside relationship between the software entity and the entities external to that software with which it must interact. The context diagram is offered as an intermediary view that will lend some discipline and order to the identification of use cases. We will identify one or more use cases for each terminator and perform a dynamic analysis on each of those use cases.
For each terminator in Figure 20, identify one or more use cases through which the intended functionality will be accomplished identifying the actors deriving benefits from the system to be created. One use case may expand into several extended or included use cases to cover a more complex situation. You will note that the opening gambit has been arranged to provide structure in the identification of needed use cases. Next, for each use case, build one or more scenarios dealing with how that use case interacts with the system. These scenarios can be in text form, a list of events, or some kind of diagrammatic treatment.

Then, express each scenario in the form of an activity and/or sequence diagram cast at the UML entry level initially. The activity diagram can be thought of a functional flow diagram similar to that used by software programmers many years ago or by system engineers in traditional structured analysis. It may be drawn vertically or horizontally as far as the author is concerned. Swim lanes may be overlaid upon the activity diagram each of which corresponds with the lower tier entities (classifiers in UML) that will be responsible for implementing activities within their swim lane. This is a key point in the analysis where the analyst must make lower tier product entity structure decisions that should lead to adding software entities to the product entity structure. Some analysts prefer to think of the two-dimensional artifacts in the diagram as states rather than activities or functions. Simultaneity is not permitted in normal state diagramming but UML permits it in activity diagrams to cover decisions and branching in a way similar to that applied in flow charting.

Alternatively, the analyst can use a sequence diagram to open the exposure of the details of a use case scenario. The entities (classifiers) through which the functionality and behavior are explored are identified on what are called the life-lines drawn as dashed lines below selected physical or
Each of these diagrams (activity and sequence) identify a lower tier (white box) view of what entities will be needed to accomplish the exposed functionality and behavior, what will have to happen in the system in order to achieve the intended goals of the use case signified in its name, and offer a detailed view of the order in which these things will occur.

The analyst can allocate top-level functions (activities) to specific entities and arrange the blocks of the activity diagram in corresponding swim lanes linked to these entities. These swim lanes will correspond to nodes, or even higher-level entities, at the system level but in any case we can refer to them in general as physical or logical classifiers within UML. If appropriate, analyze each of these classifiers from a dynamic perspective using some combination of sequence, communication, activity, and/or state diagram. The communication diagram shows the same information as a sequence diagram with an emphasis on the entities rather than the relationships between those entities. A state diagram is useful where there exists some finite number of possible conditions within which the entity can exist and there appear to be understandable rules for the entity to change from one condition (state) to another.

Identify requirements derived from these artifacts and capture them in a program-wide RAS linked to the modeling artifact (MID) that encouraged their identification and the physical or logical classifier, referred to more generally by a product entity ID (PID) in the RAS, that will be responsible for responding to that requirement and in the specification for which it should reside.

2.2.2 The Connection Between Modeling Artifacts, Specification Content, and Product Entities

Figure 22 suggests a hierarchical relationship between the elements of the UML analysis and offers a way of assigning MID. The capabilities in the specification format (template Paragraph 3.2) coordinate with terminators, use cases, extended use cases, and/or scenarios. Figure 22 only shows one terminator expanded but the intent is that for any top level software classifier AX (highest tier SW entities) entered into the product entity structure, one or more terminators would be identified and expanded as shown in the one case shown in Figure 22. The software top-level software classifier, AX, is, of course, a member of the product entity structure (PID) where X is a string of base sixty or decimal delimited base ten numerals. The suggested UML MID stream is identified first with a unique UXh MID (with \( h = e\{1, 2, 3\} \) in the example shown in Figure 22) for each terminator.

The terminator MID can be further decomposed using the MID UXhijk pattern composed from a set of use cases, possible extended use cases, and scenarios for each terminator h. These MID are the entries to place in the RAS for software requirements derived from these artifacts. In general, capabilities will be derived from these artifacts and the requirements subordinate to them will be derived from the dynamic modeling artifacts UXhijk1 through UXhijk4.
So finally, the requirements for each capability flow out of applying the subordinate UML dynamic modeling artifacts. As in TSA with lower tier product entities identified from lower tier function allocation, the lower tier software product entities flow from the sequence (life-lines) and/or activity (swim lanes) diagramming. This is a particularly satisfying coordination between lower tier entities being exposed through functional analysis in TSA and activity analysis in UML where both are using essentially the same model to identify lower tier entities.

We can continue to apply UML in the lower tiers as covered in Figure 21 treating each classifier as a system in accordance with the steps discussed above progressively identifying nodes, components (possibly in more than one layer), and classes. If the system level problem space was entered using UML as this process moves forward and downward, it will be decided that some classifiers will be developed as software and hardware entities, with the latter possibly splitting in lower tiers into hardware and software entities. The continuing analysis of hardware entities can be accomplished using traditional structured analysis or SysML model artifacts while the modeling of software entities continues primarily using UML model artifacts. In that it is
difficult for software to contain hardware, the use of TSA as the entry analysis probably makes more sense.

At the lower tier UML analyses where the physical and logical classifiers are classes and objects, the lines that flow between the classifiers on the corresponding sequence and communication diagrams coordinate with messages and influences applied to/from those classes in relation to external physical and logical classifiers (other classes generally at this level). This same effect is in operation whether the classifiers depicted are classes, components, or nodes. Each classifier must have identified for it one or more operations (services or functions) that it performs relative to an outside set of actors and one or more data entities it deals with internally to accomplish those operations. The data elements will flow into the classifier via the lines on the sequence and communications diagrams and data may be created or altered internally. Initially, the analyst may choose to first identify node, component, class responsibilities and subsequently as the analysis matures translate these responsibilities into operations and attributes.

In lower tier analyses, the assigned IPPT are responsible for identifying and defining needed interfaces below the level initially identified by the PIT. Where the two terminals of an interface touch only entities that are the responsibility of a single team, that team is clearly responsible for interface identification, definition, and integration. Where an IPPT is responsible for only one terminal of an interface, that team must cooperate with the team responsible for the other terminal to develop that interface. The integration agent in this case is the lowest common team. If there is only a single layer of teams under the PIT, the PIT is always the lowest common team. In general, it should be the team on the receiving end of an interface that first identifies the need for a new interface since interfaces should not be defined based on what is available but what is needed. If it is not obvious what team shall be responsible for the new interface, the PIT shall act as the integration agent until such time as the source is identified and then the lowest common team will take over that responsibility.

As IPPT identify lower tier entities during use case analysis, the PIT shall approve those additions and assemble them into the formal product entity structure. These entities may be physical or logical entities but eventually all of them must be identified as real product entities that will be developed. These final entities can be logical entities as in the case of computer software that will run on a particular hardware computer entity. Where it appears that lower tier entities will entail significant complexity, new IPPT may be formed by the PIT that will be subordinate to the appropriate existing IPPT. Those lower tier IPPT will take over the continuing analysis of use cases appropriate at that level and the immediately superior team will take on the role of the system engineer for the new team just as the PIT does for all top tier IPPT.

The use case analysis process employed by an IPPT or a collection of teams will necessarily be a collaborative process involving people from several different specialty disciplines. Each such team will have a leader whose responsibility it is to bring the analysis to a conclusion as scheduled. These teams will come into being, exist for a brief period of time, come to a conclusion, and pass from the scene with others replacing them. Once a use case analysis has been completed by a team, the work products will have been captured in modeling applications and integrated relative the existing work products. The modeling front will move down through the advancing product entity structure till the system is fully characterized.
Where personnel from other teams are required to accomplish work on another team's use case analysis, the owning team will cover the manhours of all personnel working the use case. Where all team members are physically collocated in the same facility, they may be depended upon to interact well through a traditional meeting in the common facility. Where at least some of the people required on a use case analysis effort are not collocated, it will be necessary to extend the meeting place geographically through the use of a product such as webex where people from several different geographical locations can cooperate in the development of a set of information.

Leaders of use case analysis teams are responsible for the prompt completion of team activity with good results but in so doing they will be well served to identify the most effective collaborative engineer on the team to lead collaborative team activity in the form of meetings especially where those meetings entail the use of distance integration aids like webex. The collaborative discussions in meetings need not necessarily be led by the team leader.

As the team completes its modeling and requirements work, the results should be reviewed by the parent team and, if approved, the team should be empowered to proceed with design at a level appropriate to the team responsibilities. The team must continue any responsibilities it may have for integration and optimization and leadership of subordinate teams that may still be involved in modeling work. This design work will entail some combination of hardware and software design development.

2.2.3 Dynamic Modeling Artifacts Explained

The use case diagram is considered a dynamic modeling artifact also but it is treated here as a transition medium between classifiers and the dynamic model set. The remaining dynamic models are implemented in four diagrams from which we may select any subset including all of them, any one of them, any pair, or any trio for a particular scenario analysis. It is not necessary to use them all for any particular analysis. Use those that make it possible to understand the problem space and properly characterize it in requirements. Some very large programs have done much of the analysis with only use case and sequence diagrams. The more complex the problem space and the more intimately the parts of the evolving system interact, the more different views of problem space that will be useful.

The first two kinds of diagrams covered, sequence and communications are both modeling the same relationships and collectively referred to as interaction diagrams. The sequence diagram emphasizes the time ordering of messages and the communication diagram emphasizes the organization of the objects that participate in the interaction. The second two are forms of state diagrams in the mind of many analysts.

2.2.3.1 Sequence Diagram

Some programs apply only the sequence diagram to explore the dynamic behavior of use cases and this may be adequate on relatively simple problems. In this tutorial we are assuming that the analyst employs the sequence diagram as the initial dynamic model, though an alternate approach would be to use the activity diagram for that purpose, but uses at least some of the
other three models to explore the use case more thoroughly. The sequence diagram example in Figure 23 illustrates the fundamentals showing two classifiers that comprise the classifier AX that has previously been analyzed. Here we conclude that AX must consist of AX1 and AX2 and that in order to accomplish the behavior defined for AX these lower tier classifiers must interact with an outside entity called an actor which will derive some kind of benefits from the relationship. The two subordinate classifiers will provide certain operations that are not clearly defined on this diagram. In the process of doing so, they will exchange messages in a certain order with time running down the page.

Each classifier including the actor has a lifeline in the form of a dashed line running down the page. A block is overlaid on this dashed line to indicate the active period(s) of classifier. Between these classifier lifelines we see messages being passed from one classifier to another. The names of these classifiers are formed of one or more words concatenated together without spaces and all but the first word capitalized. After the message name one can insert an argument list parenthetically.

The model permits the creation of classifiers and when they have performed their activity they can be killed. These features are not illustrated in Figure 23. The kinds of messages identified are: (1) a call invokes an operation on a classifier on the arrow end of the message, (2) a return message returns a value to the caller (dashed line used), (3) a send message sends a signal to a classifier, (4) a create message creates a classifier, and (5) a destroy message destroys a classifier. Message two and four could be an example of messages types 1 and 2. Message three is an example of message type 1 where the classifier is making the call upon itself.

Figure 23  Sequence Diagram Example

After a classifier sends a signal to another classifier the sending classifier returns continues its own execution. The target classifier independently decides what to do about it. A common reaction would be to trigger a state machine causing the target classifier to execute actions and change state.
2.2.3.2 Communication Diagram

In some cases we are primarily interested in the time ordered sequence of messages between classifiers but in other situations we may be more interested in the organization of the classifiers and a communication diagram can offer better results even though the sequence and communication diagrams are semantically equivalent. Figure 24 illustrates a communication diagram reflecting the same situation as Figure 23. The classifiers are joined by lines corresponding to the relationships between them and messages are related to these line each in terms of message name, direction, and the relative timing of the message.

![Communication Diagram Example](image)

Figure 24  Communication Diagram Example

2.2.3.3 Activity Diagram

An activity diagram can be used to express the things that one or more classifiers must accomplish. It is weak in terms of absolute timing of accomplishing those things but strong in expressing the relative order of those things. As shown in Figure 25, activities are illustrated by rounded corner boxes and they are connected into a sequence by directed line segments. In addition to the activities, we also will wish to show simultaneity through the use of forking and joining structures and alternative paths using branch and merge structures. The guard expressions give information about the conditions that correspond to movement through one branch or another.

This is the functional flow diagram of UML though many analysts prefer to think of the blocks as states rather than functions. The author prefers not to consider them states because it is in conflict with the notion that an entity must be in only one state at a time.

The diagram can be built with swim lanes, or not, that relate to classifiers, the same classifiers identified on sequence diagrams using the lifelines. It is through these two devices that we can allocate software functionality to classifiers. As we do so we determine the next lower tier product entity structure and should offer up newly identified entities to the PIT for inclusion in the product entity structure.
2.2.3.4 State Diagram

An interaction diagram (sequence or communication) models the relationships between a collection of classifiers while a state machine models the behavior of a single classifier. The classifier in this case can be the whole system or a classifier at any level of abstraction. The state machine models the possible condition that a classifier can exist in and the transition of that classifier from one condition or state to another based on a stimulus that might be a signal from another classifier, the passage of time, receipt of a call message invoking an operation, or a change in some condition.

A state is drawn on a state diagram, as shown in Figure 26 showing a state machine for temperature control, as a rounded corner box with the name of the state written inside. Generally the diagram must have an entry and final state symbols though it is possible that once entered the state machine may continue forever. In some cases the intent may be for the machine to continue forever, as in a traffic light system, the reality is that such a system may have to be shut down for maintenance and does need a final state. Transitions between states triggered by events in the life of the classifier being modeled are shown diagrammatically as directed line segments between a pair of state and named in a way to convey how that transition is triggered. It is understood that the machine must be in only one state at a time and that only a single transition is possible at one time. Transitions are generally thought of as taking place instantaneously.
While UML does not prescribe it, a pair of dictionaries can be helpful in clearly stating the intended operation of a state diagram. One dictionary lists the states and defines them with precision while the other lists and defines the transitions. You will note that the transitions in Figure 26 have not been named uniquely but should be in the general case so that each can one can differentiate between them in such a listing.

2.2.4 Static Entity Analysis

In early object oriented analysis (OOA) as prescribed by Grady Booch, Peter Coad teamed with Edward Yourdon, James Rumbaugh, et. al., and many other practitioners, the proper way to enter problem space was using the static view with objects. Then they encouraged the analysis of the objects from a dynamic perspective with data flow diagrams for functionality and state diagrams for behavior. This approach is possible with UML using the foursome of dynamic modeling artifacts discussed in the prior section and it can be effective when developing a largely preceded system that can be observed in the real World like a new payroll system. The author believes that largely unprecedented problems are best attacked using Sullivan's encouragement of form follows function leading with the dynamics views and identifying the static entities that populate the product entity structure from a software perspective.

UML identifies three levels of static entities but they are all product entities and while drawn on modeling diagrams using different images, they are essentially the same at different levels of abstraction. All are simply illustrated on the system product entity diagram illustrated in Figure 14 as blocks. Figure 27 illustrates the three static entities collectively referred to as classifiers in this tutorial.
In this tutorial the case is made for first identifying the nodes which are entities that will be associated with run time software. They are higher-tier entities. Like classes and components, they have associated attributes and operations. They interface with each other and possess lower tier interfaces between components and classes that comprise them. The nodes are identified in the dynamic analysis of the top-level software entity by identifying sequence diagram lifelines and activity diagram swim lanes. We then analyze these nodes from a dynamic perspective and identify components in the same fashion.

The alternative approach first identifies classes corresponding to observable entities in the problem space which are then dynamically analyzed leading to an understanding about how best to package these entities based on collecting the classes with the most intense interface relationships together as components. Just as in the use of interface analysis in TSA to validate the product entity decisions in functional analysis by observing possible unintended interface intensities, we can in UML re-consider the particular swim lanes and lifelines we selected in the dynamic analysis.

In this section, the intent is to explain what the UML static entities are and how they are used on diagrams. We will use classes in order to do so with the understanding that nodes and components are but higher tier classes. First we will describe a general class then explore structural relationships between these classes and finally we will cover the messages that are passed between them.

2.2.4.1 The Class

A class is illustrated as a box as shown in Figure 27c. The name of the class is placed in the top portion of the box, attributes are listed in the middle portion of the box, and operations are listed in the lower portion of the box. A forth box can be included below the operations in which we inscribe class responsibilities in free-form text. A responsibility is a contract or an obligation of a class. You may find it useful to begin the analysis of classes this way translating these into attributes and operations that best fulfill the class’s responsibility as the model is refined.
2.2.4.2 Class Relationships

Figure 28 illustrates the structural relationships recognized between classes. A class is said to be dependent on another if it depends on that other class for information or services. A class can be linked hierarchically to another through a generalization. Class associations can have the three adornments illustrated in Figure 29.

NameOne is the base class. NameTwo and NameThree are leaf classes in a generalization. NameOne depending on NameFour for information and services.

An association is a structural relationship

- Association Name
  NameOne \( x..y \) NameTwo

- Association Role
  The face that the class at the far end of an association presents to the class at near end of the association. Role names called end names.

- Association Multiplicity
  Tells how many objects may be connected across an association instance. Given by a range of numbers.

- Association Aggregation
  Expresses a whole-part relationship between to associated classes.

Figure 28 Structural Relationships

Figure 29 Association Adornments
2.2.4.3 Messages

Classes can also be related through the five kinds of messages discussed in Paragraph 2.2.3.1. Such a message can convey to a class a variable argument (value) that is needed in a class operation or it can convey a signal that causes the class state machine to transition to a new state for example. The messages that must be passed between classes are understood in the context of the sequence diagram under the assumption that the dynamic analysis is accomplished prior to the static analysis.

2.2.5 Related Analyses

2.2.5.1 Specialty Engineering

The specialty engineering matrix discussed in paragraph 2.1.9 can be used in software as well as hardware to identify all product entities for which specialty engineering requirements must be developed. This includes, for example, safety, reliability, and security. The software interface requirements fall out of the sequence and communication diagram analyses and flow into the specification template offered in Figure 27.

2.2.5.2 Environmental Requirements

Software environmental requirements are somewhat different from the hardware and system environmental requirements that tend to be dominated by the natural environmental factors. The software being an intellectual entity rather than a physical one, is shielded from the natural environmental stresses. True, the software operating within a machine that can suffer adverse environmental stresses can as a result fail, but this is a secondary effect. Reasonable software environmental relationships include any language restrictions and the computer architecture upon which it must run, for example.

2.2.6 Specification Structure

The specification outline offered in Figure 5 can also be applied to software entities where the capabilities are linked to either the terminators, use cases, extended use cases, or scenarios and the subordinate requirements listed under each capability are drawn from the corresponding dynamic diagramming (activity, sequence, communication, and state diagramming) work. Thus the requirements can be clearly shown to trace to modeling artifacts just as the performance requirements in hardware can be shown to flow so clearly from functions. Figure 30 offers an outline for a software requirements specification (SRS) using an edited template from EIA J STD 016 to integrate the supporting modeling work into the specification. Note the similarity to the outline in Figure 5 for a system or hardware item performance specification.
<table>
<thead>
<tr>
<th>PARA</th>
<th>TITLE</th>
<th>MODEL</th>
<th>DEPT</th>
<th>APP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>SCOPE</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>2.</td>
<td>APPLICABLE DOCUMENTS</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.</td>
<td>REQUIREMENTS</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1</td>
<td>Required states and modes</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.1</td>
<td>Classifier context diagram</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.2</td>
<td>Use case analysis</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.2.h</td>
<td>Terminator h</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.2.h.i</td>
<td>Terminator h, use case i</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.2.h.i.j</td>
<td>Terminator h, use case i, extended</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.2.h.i.j.k</td>
<td>Terminator h, use case i, extended use case j</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.3</td>
<td>Dynamic Analysis</td>
<td>DID</td>
<td>2100</td>
<td>D</td>
</tr>
<tr>
<td>3.1.3.h</td>
<td>Terminator h dynamic analysis</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.3.h.i</td>
<td>Use case hi dynamic analysis</td>
<td>UML</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.3.h.i.j</td>
<td>Extended use case hij dynamic analysis</td>
<td>UML</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.3.h.i.j.k</td>
<td>Scenario hij dynamic analysis</td>
<td>UML</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.3.h.i.j.k.1</td>
<td>Sequence diagram hijk1 analysis</td>
<td>UML</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.3.h.i.j.k.2</td>
<td>Communication diagram hijk2 analysis</td>
<td>UML</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.3.h.i.j.k.3</td>
<td>Activity diagram hijk3 analysis</td>
<td>UML</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.3.h.i.j.k.4</td>
<td>State diagram hijk4 analysis</td>
<td>UML</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.4</td>
<td>Lower tier classifier identification</td>
<td>UML</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.2</td>
<td>Entity capability requirements</td>
<td>UML</td>
<td>2100</td>
<td>A</td>
</tr>
<tr>
<td>3.2.m</td>
<td>Capability m</td>
<td>UML</td>
<td>2100</td>
<td>A</td>
</tr>
<tr>
<td>3.2.m.n</td>
<td>Capability m, requirement n</td>
<td>UML</td>
<td>2100</td>
<td>A</td>
</tr>
<tr>
<td>3.3</td>
<td>Interface requirements</td>
<td>UML</td>
<td>2100</td>
<td>D</td>
</tr>
<tr>
<td>3.3.1</td>
<td>External interface requirements</td>
<td>UML</td>
<td>2100</td>
<td>D</td>
</tr>
<tr>
<td>3.3.1.m</td>
<td>External interface m</td>
<td>UML</td>
<td>2100</td>
<td>D</td>
</tr>
<tr>
<td>3.3.1.m.n</td>
<td>External interface m, requirement n</td>
<td>UML</td>
<td>2100</td>
<td>D</td>
</tr>
<tr>
<td>3.3.2</td>
<td>Internal interface requirements</td>
<td>UML</td>
<td>2100</td>
<td>D</td>
</tr>
<tr>
<td>3.3.2.m</td>
<td>Internal interface m</td>
<td>UML</td>
<td>2100</td>
<td>D</td>
</tr>
<tr>
<td>3.3.2.m.n</td>
<td>Internal Interface m, requirement n</td>
<td>UML</td>
<td>2100</td>
<td>D</td>
</tr>
<tr>
<td>3.4</td>
<td>Specialty engineering requirements</td>
<td>DID</td>
<td>2100</td>
<td>E</td>
</tr>
<tr>
<td>3.4.m</td>
<td>Specialty Engineering Discipline m</td>
<td>Specialty Scoping</td>
<td>E</td>
<td></td>
</tr>
<tr>
<td>3.4.m.n</td>
<td>Specialty Engineering Discipline m, Requirement n</td>
<td>Specialty Scoping</td>
<td>E</td>
<td></td>
</tr>
<tr>
<td>3.5</td>
<td>Environmental conditions</td>
<td>3-Layered Env Model</td>
<td>2100</td>
<td>B</td>
</tr>
<tr>
<td>3.6</td>
<td>Precedence and criticality of requirements</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>4.</td>
<td>VERIFICATION</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>PACKAGING</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>6.</td>
<td>NOTES</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>6.1</td>
<td>Requirements traceability</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>6.1.1</td>
<td>Inter-specification traceability</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>6.1.2</td>
<td>Verification traceability</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>6.1.3</td>
<td>Modeling traceability</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>6.1.4</td>
<td>Section 2 traceability</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>6.1.5</td>
<td>Programmatic traceability</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>6.2</td>
<td>Glossary</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>6.3</td>
<td>Specification maturity tracking</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
</tbody>
</table>

Figure 30  Software Requirements Specification (SRS) Template
Refer to Exhibit B of this student manual for a JOG System Engineering data item description (DID) that shows how the subparagraphs of paragraph 3.1 relate to the subparagraphs of paragraph 3.2. Paragraph 3.1 essentially provides an opportunity to capture the complete UML analysis for the classifier covered in the specification. This can be done by actually including the diagrams discussed in prior paragraphs in this text or by referencing them in a computer application within which the modeling is done and derived requirements captured or the diagrams can be completed manually (or with a computer graphics application like Microsoft Visio or Powerpoint) and contained in the appendices of the system definition document (SDD).

The reader will note that in Figure 19 there is a second plane labeled UML for the case where the program chooses to capture the UML analysis work products in the SDD. On a program that is dealing only with a UML analysis for a software product the Appendix structure shown in Table 1 column A could be used. If the program must deal with both TSA and UML, the software appendices could be simply added to those required for TSA and lettered G through N with the TSA Appendix G (the RAS that should also contain the software requirements derived from UML artifacts using the MID pattern illustrated in Figure 22) becoming a common RAS in Appendix O. The latter pattern is captured in column B. In this combined case, the classifier diagramming can remain in Appendix N but Appendix C should capture the aggregate product entity structure including hardware and software entities.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>MODEL ARTIFACT</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>G</td>
<td>Context Diagrams</td>
</tr>
<tr>
<td>B</td>
<td>H</td>
<td>Use Case Diagrams</td>
</tr>
<tr>
<td>C</td>
<td>I</td>
<td>Scenarios Text</td>
</tr>
<tr>
<td>D</td>
<td>J</td>
<td>Sequence Diagrams</td>
</tr>
<tr>
<td>E</td>
<td>K</td>
<td>Communication Diagrams</td>
</tr>
<tr>
<td>F</td>
<td>L</td>
<td>Activity Diagrams</td>
</tr>
<tr>
<td>G</td>
<td>M</td>
<td>State Diagrams</td>
</tr>
<tr>
<td>H</td>
<td>N</td>
<td>Classifier (Object, Class, Component, and Deployment) Diagrams</td>
</tr>
<tr>
<td>I</td>
<td>O</td>
<td>RAS</td>
</tr>
</tbody>
</table>

The reader should note that in software using UML it is possible to relate the modeling machinery very clearly with the classifiers about which specifications are to be written. It is not quite so easy in hardware using TSA because all of the performance requirements derived from a particular function may be allocated to the same entity. It might be possible to force fit the functions into a closer alignment with the product entity structure by restarting the functional analysis for each entity identified but the author does not think this would be particularly helpful.

2.2.7 Software Requirements Close-Out

When a set of classes that collectively form a component have been thoroughly characterized including the sets of requirements associated with those classes and dynamic relationships and the related requirements have been captured, reviewed, and approved, the responsible team can
start to develop the computer code that will implement the component. This process can begin across the lower tier of the components and continue to next assemble into computer code for the nodes. This process may take place in a single string or simultaneously following the same pattern in multiple paths where the software is distributed to some extent. There is, of course, a continuing need for integration and optimization across the development being accomplished by different teams that focus primarily on interfaces just as in hardware development.

The PIT and program manager will reach a conclusion for the program with possible local differences whether to pursue software design and coding in a top-down or bottom-up fashion. In either case, any code developed must be tested at each level of build. This may require the development of special stub (top-down) and/or drivers (bottom-up) to permit testing of completed portions before higher or lower tier software has been completed.

The classifiers identified in the UML analysis should be entered into the product entity structure by the PIT so that a consolidated view of the overall system matures. Figure 31 shows the product entity structure evolving based on feed from all modeling activities being applied on the program under the watchful integrating eye of the PIT.

![Figure 31  Evolving Product Entity Structure](image)

The UML analysis process encouraged does involve a partitioning of the whole problem space into parts hierarchically arranged and assigned to teams. These teams, in hardware and software, can generally be depended upon to integrate and optimize at the level for which they are responsible rather than the next higher level. Therefore the PIT (and superior IPPT) must integrate and optimize across the best efforts of the teams. The PIT should also consider special integration studies across the grain of the system based on overarching functions such as shutdown and activation as well as others that may be suggested through system level states and modes analysis. These efforts should be led by the PIT and participated in by people from a wide range of teams.

2.3 Opening the Analysis With DoDAF

DoDAF was developed to support modeling of the complex information systems that DoD has been assembling in recent years to interconnect their many sensor and weapons systems to form
an effective military capability often referred to as sensor-to-shooter connectivity. It does not provide a complete modeling set primarily because of the absence of a model artifact for the physical product entities. The product entity view can be provided by augmenting the DoDAF model set with the TSA product entity diagram, however, from the aggregate model set this description assembles. Physical products identified in the DoDAF process can be inserted into the TSA product entity diagram just as they can be using any of the model sets and as suggested in Figure 31.

The DoDAF problem space entry is similar to the UML entry and in fact can use some UML artifacts to do so. DoD does not prescribe any particular set of modeling artifacts for the several views. In this paper we will use a combination of UML, IDEF, and TSA artifacts. DoDAF employs four modeling views: (1) two all view products that offer a textual overview of the system (as in what was once called a mission need statement) and an overall glossary explaining all terms used; (2) two technical view products that define standards that must be respected on the program and the evolution of those standards over time; (3) nine operational view products that capture how the user views their needs; and (4) thirteen system view products that offer the engineering perspective on the needed system. Figure 32 illustrates a recommended DoDAF development sequence. The simulation work could better be shown overarching both the analysis and design work interacting with and serving both. Note that the two all and two technical views would be a good addition to any modeling apparatus.

Figure 32  DoDAF Development Sequence

The technical and all views work products should be developed initially by the user and acquisition agents and evolved by the contractors selected to develop the system. The user should have developed the operational views in the process of evolving a set of documents used to refine their need. These documents include a Joint Capabilities Document (JCD that takes the
place of the old need statement cut very broadly. In the study process prescribed by DoD for a new system, the capability identified in the JCD may be cut up into n systems each of which should have developed for it an Initial Capabilities Document (ICD) that is the equivalent of the old mission needs statement for a given system to be procured. The ICD is then matured into a Capabilities Development Document (CDD) and eventually a Capabilities Production Document (CPD). In between these last two a contract would commonly be let resulting in the development of a system specification attached to the contract. DoDAF can be used to gain a systematic insight into the content needed in all of these documents.

2.4 Integrated Modeling

A program must make a decision about the modeling techniques it will apply as it builds the proposal. This may include some mix of TSA, UML, SysML, DoDAF, and IDEF-0. A program that is going to be primarily computer software or networked assets could enter the program with UML or DoDAF. If the product is a database, it could enter with IDEF-0 or UML. But, generally, modeling entry should involve the use of TSA or SysML at the system level because software, an intellectual entity, must run on hardware entities that provide the product with real substance. As noted earlier, at the time this was written SysML was not yet fully ready for use so TSA would be the author's preference now. But, an enterprise should continue to follow the development of SysML and work toward replacing TSA with SysML. In any case, there is a need to recognize that for some period of time there will be a need for a model that works well for systems and hardware and another model that works well for SW. For now, also, there is not one great computer application within which one can model HW and SW requirements work and permit easy cross model traceability and provide specification publication capability so it will be necessary to use two or three applications to cover the needed tool set.

Work can be accomplished well for systems and HW as covered under paragraph 2.1 and for SW using the approach covered under paragraph 2.2. When applying both, problems will tend to arise when transitions have to be made between these two approaches. It is not possible for SW to include HW but the opposite case is perfectly normal. So, the transitions will only be a problem as the analysis shifts from HW to SW moving from the use of TSA or SysML to UML. There are two concerns at this point: one in the models applied and the other in the computer applications employed.

The transition point will occur when the highest tier software entities are identified. There may, of course, be several of these transitions distributed about the expanding product entity structure. The program has the option of pooling all of the software into an integrated entity or permitting it to be distributed within multiple processors that may still all be under the responsibility of one team or distributed among teams with both hardware and software responsibilities. If we can solve one of these hardware-software handoffs we will have solved the general problem of requirements traceability across these gaps.

It should be clear that requirements traceability to models is assured in the approach covered in this tutorial because all of the requirements are to be derived from a model. Vertical or hierarchical requirements traceability is very simple in specialty engineering areas in hardware, software, and across the gap as described in this tutorial. The environmental requirements are
vitally different between hardware and software and one can make a case that lower tier software environmental requirements should not have to respect traceability across the gap to higher tier hardware or system environmental requirements that are largely environmentally related. Precisely the same method of identifying hardware interface requirements can be used to identify software interfaces as well as hardware-software interfaces because we identify them between entities that appear in the joint product entity structure. So, if interface requirements traceability involves lower tier interface expansion requirements to higher tier interface requirements, traceability is assured. This leaves only the performance requirements a remaining problem from a vertical or hierarchical traceability perspective.

Given that the system entry analysis was accomplished in TSA using some form of functional analysis and the lower tier software analysis is going to be done using UML, there is a temptation to employ activity diagrams in UML to analyze software entities from a dynamic perspective because it is very similar to functional flow diagramming and might give us some interesting opportunities to link up hierarchical traceability. However, for a given software entity there may have been 10 performance requirements derived from 8 functions allocated to the software entity in question. There is no clear way to link up the activity analysis and requirements derived from it with the several functional analysis strings and the performance requirements derived from them that can easily be automated.

So, let us pursue another tack in an attempt to coordinate the traceability relative to the sequence oriented dynamic analysis approach described previously. If requirement R%1 is one of a set of requirements R% through R%10 where R%1 is derived from function F#1 of a set of functions F#1 through F#8 and requirement R%1 is allocated to product entity AX2 and it is decided that AX is going to be developed as a software entity, then one of the scenarios to be analyzed will be UXhijk. Assume that we accomplish the dynamic analysis using sequence diagram UXhijk1 from which we derive requirement R@1. What we are looking for is a way to establish hierarchical traceability between requirement R@1 and some requirement in the set R% through R%10. The X, %, and # characters are being used to designate base 60 strings in this discussion. We know that requirement R@1 must be traceable to one of the 10 performance requirements allocated to classifier AX2 and we can look at that list of requirements and select the one most closely related.

To make this selection more organized, we can form an x by y matrix, in this case a 10 by 12 matrix, and pair-wise compare the sets R@ and R%. In Figure 33 you can see this whole process taking place The 10 functionally derived requirements are captured in the RAS mapped to the set of functions F#1 through F#8 and allocated to product entity A&. Based on these requirements we build a context diagram for entity A& and analyze A& from the perspective of each of the three terminators shown. As an example Use Case U&3 is extended to three use cases and we build three scenarios one of which, U&3111 is analyzed from a dynamic perspective with some combination of sequence, communication, activity, and state diagrams. Requirements R@1 through R@12 are derived from these analyses and captured in the RAS (possibly linked to the RAS database from a UML modeling application).

There are 10 requirements (R% through R%10) to which the requirements in the set R@1 through R@12 will have to hierarchically trace. We can build a 10 x 12 matrix and pair-wise analyze the
relationships between the two sets of requirements, perhaps concluding that one of the matches is \( R@_i \) traces to \( R%_j \). All of the matches are marked in the requirements management database table for traceability relationships. There are no known databases that provide the traceability evaluation matrix so it may have to be accomplished as a pencil and paper aid. However we should be able to set the requirements management database filter for the two sets of requirements of interest aiding in the identification of the sets of interest for a particular case. In this example, there might be ten or more sets of requirements like \( R@ \) derived from ten or more dynamic analyses. In each case an \( x \) by \( y \) matrix would be needed to pair-wise analyze the traceability relationships. In any case, it should be clear that we can have good requirements to modeling traceability and even good hierarchical traceability across the gap between performance requirements.

![Requirement Traceability Matrix](image)

Figure 33  Requirements Traceability Across the Gap

In both TSA and UML we have discussed a decomposition process that partitions the problem space into parts in which the analysis is accomplished. Whenever we partition any whole we have an obligation to integrate and optimize across the boundary conditions thus created. The PIT must accomplish this integration work relative to the top level IPPT and each IPPT with lower tier teams must accomplish this work relative to it's own immediately subordinate teams. Much of this integration work will take place at the interfaces ensuring that requirements on one end of an interface are compatible with those for the other terminal. Each team with subordinate teams, however, should also integrate across its immediately subordinate teams relative to the requirements derived at the subordinate team level relative to those at the parent team level. Part of this work can be accomplished by simply establishing the traceability between the requirements at the two levels. Another approach of value is to accomplish higher tier function effects across the lower tier team responsibilities. For example, one can inquire collaboratively into lower tier performance of higher tier functions like turning the system or entity on or off,
moving from one major mode to another, accomplishing some kind of transfer function, or physical separation or joining of two entities.

Another kind of traceability can also be used to stimulate integrating results. This was pointed out to the author by an engineer at Puget Sound Naval Base in Bremerton, WA. Given a requirement at level m, we can inquire if the intent of the requirement was fully implemented in the requirements for the n entities at level m+1 (downward). This kind of traceability inspection must await the development of the subordinate specifications, of course, as does all hierarchical traceability.

At one time, in the 1950s when software was a very young discipline, it happened that hardware and software analysis, to the extent that it was done, was accomplished using exactly the same model, flow charting as shown in Figure 34. Over time, probably encouraged by the ease with which flow charts could be outputted onto line printers using ASCI symbols, computer software people got into the habit of building flow charts in the vertical rather than the horizontal axis still used by system engineering in their functional flow diagrams. The activity diagrams of UML still reflect the vertical orientation but it is really of little significance which orientation is used. The absolutely fascinating approaching reality is that system and software people will rejoin the same house in the near future. As UML and SysML become more fully integrated as suggested in Figure 34, we will achieve a tremendous milestone of universal unified modeling capability.

As we pass through this door into a world of integrated modeling and supporting computer applications, we will find it a more reasonably affordable task to integrate across the hardware-software boundary than has been the case for many years. But then as now, integration takes place in the minds of the system engineers working on the program. These engineers must be
every vigilant for inconsistencies between information sets that signal that two different domains are not working from a common understanding of the problem and solution spaces.

Figure 35  The Approaching Merge

3  Requirements Management

3.1  Summary of Team Activity During Requirements Work

During initial product development work, the PIT will model the problem space using a predefined set of modeling methods selected from the list of enterprise-approved modeling methods and apply those methods to identify top-level system entities and interface relationships and their requirements. This level of system definition shall be completed before program level IPPT are initiated on the program and staffed. These top-level product entities are the basis for assignment of these teams. When a team is established, the leader shall be presented with the specification for the top-level entity for which the team shall be responsible, a design concept (in particular if it is HW or SW), a clear definition of all external interfaces, and the corresponding components of the WBS, SOW, IMP, and IMS. The team will be charged with the development of that item and all subordinate entities and interface relationships. In the process of so doing, the team may conclude that lower tier teams are required and must request that action of the PIT.

The work products from the IPPT will be loaded into the computer applications used on the program by PIT and checked for consistency. The PIT shall perform integration and optimization work on modeling work contributed by the IPPT fitting the work products into a coherent system analysis of the problem space. As part of this work where the product is to be implemented in SW, the responsible team will seek to identify all needed use case analyses and assign them for completion by people on the team if possible. Where this work must involve people from other
teams, the responsible team must request help from the PIT and a cross team analysis effort will be established. The responsible lead team must ensure that each analysis is complete with all needed supporting modeling work.

The PIT shall maintain the product entity structure, the interface relationships, and all requirements modeling and management assets. The requirements shall be retained in a relational database from which specifications may be printed to paper or computer screen and within which traceability may be maintained. This database shall be linked to one or more modeling applications used on the program. The modeling applications shall be used for the purpose of identifying the system entities, interfaces, and appropriate requirements in each case. All content of these applications shall be under the control of the PIT until such time that it is formally approved at which time it shall fall under configuration management control and shall not be changed without a formal approval as well. Responsibility for data entry can be distributed to IPPT or retained by PIT. Entry may be aided by special Microsoft Office applications making it unnecessary for personnel to develop and maintain computer application skills.

The PIT will seek to establish IPPT overlaid upon the product entity structure so as to minimize the interface relationships between the entities for which the teams are responsible. The purpose of this arrangement is to minimize the need for cross team coordination and staffing for use cases analyses.

The preferred lower tier HW development approach entails a continued application of TSA using the same modeling database application applied at the system level. The preferred SW product development approach entails PIT and IPPT application of unified modeling language (UML). In the near term traditional structured analysis (TSA) will be applied in combination with UML maturing to a combination of UML and SysML as the latter matures into a formally released model by the Object Modeling Group. TSA or SysML are to be used initially to identify system and top level product entities that will more often be hardware end items. As the analysis proceeds downward and identifies a need for computer software, the analysis should switch to UML.

3.2 Requirements Tools Base

Figure 36 illustrates the preferred tools environment for programs. A requirements management database is used to capture all of the requirements that will be published in specifications and those specifications may be published from this database. In Figure 36 this is referred to as a big dumb database with no slur intended for the makers of tools that do not include integrated modeling capabilities. This is a relatively simple relational database application that can contain text information in a tabular structure. Each table row corresponds to a unique requirement with data captured in table columns for the several fields needed. Additional relational tables may be required for vertical traceability, verification traceability, and lateral (to models from which the requirements are derived) traceability. The program may use available modeling applications for UML (such as Rational Rose) and TSA (such as CORE) and arrange for traceability between these applications and the management application in an application like DOORS.
Many enterprises find it difficult for engineers to maintain currency with a set of requirements database applications as well as other applications more directly related to their work. All or most of these engineers will, as a function of accomplishing their normal work, maintain proficiency with the three fundamental Microsoft Office applications (Word, Powerpoint, and Excel). Loader applications crafted from Microsoft Office applications may be used to permit all engineers to enter data into the requirements database suite without a need for the engineers to become intimately familiar with these applications.

The PIT must exercise integration and optimization control over the requirements application suite and will require some members who really understand the applications, how they work, and how their content is inter-related. The suit must be set up to permit passing control of approved content to configuration management while retaining control of all in-work data.

3.3 Recommended Specification Responsibility Pattern

In the author's view, a program should staff a program integration team (PIT) that should begin the requirements analysis process at the system level and develop the top level diagrams in the SDD. This work should continue as necessary to develop the content of the system specification and the specifications corresponding to the top-level teams. The structured analysis for each of these teams should be taken over by the corresponding IPPTs in each case until they have completed the content of the specifications that define the problem for any subordinate teams. If no subordinate teams have been identified then they would have to complete the analysis needed to develop all of the specifications subordinate to their top-level specification. This same pattern
carries down to the lowest level. Each team should act as the system agent for all of its lower tier teams and principal engineers. This starts at the PIT for the system and works its way down through the lower tier teams. The Program Manager and Chief Engineer/PIT Manager should review and approve the system specification and all top-level specifications. PIT should establish rules for review and approval of lower tier specifications created by the teams.

With different parties doing the structured analysis, it is necessary to apply process integration and the PIT should do that accepting data into the several appendices of the SDD, numbering the figures, and cross-checking the data submitted. At least one team will be involved in software development and if traditional structured analysis has been applied for the system level, then that or those teams responsible for software will want to switch to some form of software modeling such as UML. Regardless of the modeling methods applied, all of the requirements modeling artifacts should be captured in the SDD either in the paper appendices noted earlier or referenced in the database systems used. The integrated RAS should be implemented in the big dumb database of Figure 36 as simply the basic relational database table used to capture requirements.

3.4 Requirements Risk Management

The principal risks that appear during the requirements development work involve a sensed inability to satisfy the requirements. The risk may be motivated by the conclusion that insufficient financial, or schedule resources have been made available. The concern may be that the requirement simply cannot be satisfied with available technology. Finally, the concern may be motivated by the conclusion that the value is simply too hard to achieve with available skill and knowledge. It is not uncommon to partition all into the categories of cost, schedule, technology, and performance as a result.

3.4.1 Requirements Validation

EIA 632 identifies an activity called requirements validation where we make an effort to determine to what extent we can satisfy particular requirements. The simplest way of reaching this conclusion is to simply ask the person(s) responsible for accomplishing the related design work if they can satisfy the requirements. If there is a lack of confidence then we need to proceed further in our efforts to identify potential performance risks. As requirements are identified and written we should validate them at that time. In most cases, the conclusion will be that there is no problem. Should we conclude that either there is a problem or we are not certain that can satisfy the requirement the first alternative investigated should be to ask if the requirement can be changed making it more certain that it can be satisfied. If that is not possible, then we should choose a means to mitigate the risk through an appropriate analysis, development evaluation test, simulation, or demonstration. If we believe that the requirement is very important in the development effort and that it will require some time to reach a final conclusion, we may select the requirement for more intense management as a technical performance measurement (TPM).

Parameters are managed through TPM by placing them on a list and assigning each TPM to a specific engineer who is charged with closing the gap between the required value and the currently demonstrated capability. The parameter principal engineer must maintain two charts: (1) a parameter chart that tracks these two values over time annotated with notes citing important
events coinciding with significant changes in the relationship between the values and (2) an action plan stating what is going to be done, when, and to what anticipated effect.

3.4.2 Margins and Budgets

Every program manager will experience difficult problems each requiring a tough decision. These problems can be made less severe by ensuring that the program manager has the resources available. This outcome is encouraged where the values for the most difficult requirements are combined with margins such that there is an opportunity to award engineers with a very difficult design problem some slack. These margins come in three varieties: cost, schedule, and performance. Cost margin is commonly applied as a management reserve such that the program manager can award a team more cost to solve a problem. Similarly, if a team has a design problem that can be solved through award of schedule slack time, the design may be possible. The third kind of slack is requirements margins. In all of these cases, the margin is derived by invariably making the job more difficult. Cost margin is often made available by skimming the task estimates of 10-15%. Schedule slack is obtained by subtracting available time for tasks on the critical path. Risky requirements are made more difficult to achieve. For example a weight margin may be realized by subtracting 5% from all weight figures. So the engineers will be challenged to accomplish their design with a weight value of required value - 5%. This is more difficult, clearly. The good news is that engineers will most often make these more demanding requirements preserving the margin values for the most difficult problems. When a very difficult weight problem appears, the manager can allocate some available margin. The margins invariably will be consumed before the design process is complete but there are ways of using available margins from requirements not of the same kind. For example, if an engineer has difficulty reaching his/her reliability figure after all of the reliability margin is gone, the manager can award some cost margin to permit the use of better parts or some mass margin permitting a heat sink that will reduce junction temperatures.

Requirements budgeting also has a risk reduction effect because it partitions available requirement values to the several designs at any one level of indenture. For example, given that it has been decided that 1500 watts of electrical power will be available from a source and there are 10 loads to be attached to this source, an engineer must partition this available power in a rational way between these several loads and integrate the results.

3.4.3 Risk Tracking

Risk is often measured using a dual axis criteria dealing with the probability that the concern will be realized and the degree of difficulty it will present if it does come to pass. This makes it a little difficult to track a single risk parameter over time and the way many people apply the dual axis system makes it difficult to accumulate a program metric that can be tracked over time. A variation on the safety hazard index described in MIL-STD-882 offers a way to measure risk with a single parameter that responds properly to characterize instantaneous values and a historical record for the program.

Figure 37 shows the risk matrix. Tables 2 and 3 corresponding provide the dictionaries explaining the values entered on the matrix axes. The intersections contain an index number that is simply the product of the axis numbers.
Table 2  Risk Probability of Occurrence Criteria

<table>
<thead>
<tr>
<th>CAT TITLE</th>
<th>P(O)</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>5  Nearly Certain</td>
<td>0.95-1.00</td>
<td>Will occur at least once during program</td>
</tr>
<tr>
<td>4  Probable</td>
<td>0.75-0.95</td>
<td>Will probably occur once during program</td>
</tr>
<tr>
<td>3  Possible</td>
<td>0.50-0.75</td>
<td>May occur during program</td>
</tr>
<tr>
<td>2  Unlikely</td>
<td>0.25-0.49</td>
<td>Will probably not occur during program</td>
</tr>
<tr>
<td>1  Nearly Impossible</td>
<td>0.00-0.24</td>
<td>Will not occur during program</td>
</tr>
</tbody>
</table>

Table 3  Risk Effects Criteria

<table>
<thead>
<tr>
<th>CAT TITLE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>5  Catastrophic</td>
<td>Program in jeopardy of termination</td>
</tr>
<tr>
<td>4  Serious</td>
<td>Serious damage to program</td>
</tr>
<tr>
<td>3  Moderate</td>
<td>Problems cause program focus difficulties</td>
</tr>
<tr>
<td>2  Minor</td>
<td>Problems that can be easily be overcome</td>
</tr>
<tr>
<td>1  Null</td>
<td>No problem</td>
</tr>
</tbody>
</table>
If you were to compare this information with the MIL-STD-882 safety hazard matrix, you would find that the safety hazard matrix offered in the military standard uses letters for one of the two axes and that the highest hazards (risks) have the lowest indices. Our matrix in Figure 37 uses numbers on both axes and the index values are higher for more serious risks. Therefore, it is possible to apply the index values in a mathematical sense as a program metric. Given the 6 program risks listed in the program risk list shown in Table 4 with the indicated risk index values, the instantaneous program risk index is 97.

Table 4  Program Risk List

<table>
<thead>
<tr>
<th>RISK NBR</th>
<th>RISK TITLE</th>
<th>PROB</th>
<th>EFF</th>
<th>INDEX</th>
<th>TM</th>
<th>PRINCIPAL</th>
<th>SUSP</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Life Cycle Cost</td>
<td>4</td>
<td>4</td>
<td>16</td>
<td>1</td>
<td>Burns</td>
<td>02-10-05</td>
</tr>
<tr>
<td>5</td>
<td>Payload Capacity</td>
<td>4</td>
<td>5</td>
<td>20</td>
<td>1</td>
<td>Adams</td>
<td>03-08-05</td>
</tr>
<tr>
<td>7</td>
<td>Stoddard Supplier Risk</td>
<td>5</td>
<td>4</td>
<td>20</td>
<td>3</td>
<td>Thornton</td>
<td>04-20-05</td>
</tr>
<tr>
<td>12</td>
<td>Program Funding</td>
<td>4</td>
<td>4</td>
<td>16</td>
<td>0</td>
<td>Connolly</td>
<td>03-10-05</td>
</tr>
<tr>
<td>15</td>
<td>Computer Software Schedule</td>
<td>5</td>
<td>5</td>
<td>25</td>
<td>4</td>
<td>Sampson</td>
<td>05-23-05</td>
</tr>
</tbody>
</table>

Current Program Index  97

Thus, we have arrived at a program risk index or metric. If we maintain a chart of this metric over time we see that it characteristically will rise early in a program as risks are identified but there is a delay in mitigating them causing a rising metric as shown in Figure 38. As a program progresses, this value will rise to a peak at some point in the program and subsequently start a long decline. Risks continue to be identified so we see the accumulated total number of risks continue to increase but the program is being successful in mitigating risks and later risks are commonly lower in index than those identified earlier in the program.
3.5 Verification Requirements

The author respects the six-section specification structure of the DoD specification standard where Section 3 contains the product requirements and Section 4 the verification requirements. For every product requirement in Section 3 there should be one or more verification process requirements in Section 4. Whoever writes the Section 3 requirement should also write the verification requirement and with little time between the two actions. The rationale here is that the author of an unverifiable requirement will have some much difficulty writing a verification requirement and this difficulty may stimulate them to look for better ways of writing the requirement leading to a verifiable requirement.

We commonly respect four methods of verification: test, analysis, demonstration, and examination. A verification traceability matrix should be included in every specification that correlates every requirement in Section 3 with a method of verification and one or more verification requirements in Section 4. Each one of the rows in the verification traceability matrix forms a verification string. All of the verification traceability matrices are pooled into a single program-wide verification compliance matrix shown in Figure 39 that lists every verification string.

![Figure 39 Verification Traceability](image-url)
A verification engineer or team must now assign verification task numbers to all of the strings in the compliance matrix. Each task is identified in a task matrix and coordinated with the name of a principal engineer who must plan the task, make arrangements for the needed resources in a timely way relative to the task, accomplish the task on schedule, and produce a verification report. The reports from all of the tasks for a given item may be subjected to a configuration audit by the customer to ensure that the contractor did meet all of the requirements in the specification.

3.6 Specification Review and Approval

No matter the path the specification has taken through the requirements analysis process relative to modeling, the program should pass it through a review and approval process before it becomes a part of the formal configuration baseline definition. The review and approval process, shown in Figure 40, offers a formal or informal peer review way of comparing the content of a specification with a set of standards that all specifications should meet. Following approval, the specification must be formally released, published, and made available to program personnel either in paper or on-line form. The released specification must thereafter be formally protected through configuration management. Any changes must pass back through this same process to gain approval.

![Figure 40 Specification Review and Approval](image)

The formal review process should include a conscious evaluation of specification template faithfulness and overall quality measured in accordance with a specification checklist. Next, the specification should be checked for adherence with good traceability standards. The program may choose to fully implement traceability standards shown in the figure or some subset thereof. The final string of checks shown in Figure 40 assesses the specification for residual risk, completeness and excess content. A decision is reached by the reviewers followed by the review chairman calling either for corrective action or approval of the specification, if needed.

Specifications prepared on small or advanced programs may not have sufficient budget to support a formal review process. In this case, while not as supportive of a low risk approach, a
specification can be reviewed by experienced people in a less organized fashion, called a peer review. The team is assembled and asked to review the document either together or on-line at their desk followed by a group meeting to discuss the content.

The master copy of each specification must be retained and protected by an assigned authority in order to protect the integrity of the document. Once approved and released, this master must be accurately identified and protected against change. In one organization the author recalls, the master was changed during work on an engineering change proposal (ECP) but the ECP was subsequently canceled. The organization no longer had a master for the specification in effect because it had become corrupted by the change work that did not materialize. It helps to consider each specification build or change a separate campaign that results in the release of a document that will exist forever. If subsequently that document is changed, the change is built anew on the preserved baseline past.

Specifications must be readily available to personnel working a program. As they are released they must be distributed to those who need them. As they are revised the same is true of the revisions. Years ago specifications were crafted with typewriters and type setting. These were published in paper form and distributed using shoe leather and mail systems. If most of your specifications are in paper media, you may have no near term option but to place them in a paper document library from which program personnel may check them out physically. But, even if this is the current case, you should be making plans to move to an on-line specification library for cost, efficiency, and document configuration control purposes.

In a paper media, after a specification is formally released, the master must go to reproduction where sufficient copies are made to cover the needs for distribution and the library. The master should be returned to what the author has accurately heard referred to as the vault, a physically secure facility (not in the classified data sense unless this is also a valid concern) where all of the engineering masters are retained. The copies must then be physically distributed. If the specification in question is also a customer-approved specification, another loop will be required to gain their approval prior to distribution. A networked library will avoid a great deal of this busy work.

Adios paper and good riddance. Even if you are currently using a paper media for distribution you probably already have the resources in place to convert to networked computer media. It requires specifications captured in computer file media, a network with adequate storage capacity and speed, and easy access from work stations on the part of the people. These features are present in most everyone’s shop today or are not beyond the pale to achieve. It is unimaginable that anyone would use a typewriter today to prepare a specification so they will always be created in some form of word processor or computer database. The results of this work can be passed on to the document release function on a disk or via the network connection and thereafter transferred to an on-line library from which anyone may open it but not change it.
VOLUME 2R
AN EFFECTIVE SPECIFICATION DEVELOPMENT ALGORITHM
STUDENT MANUAL

EXHIBIT A
PRESENTATION MATERIALS
AN EFFECTIVE SPECIFICATION DEVELOPMENT ALGORITHM TUTORIAL

SPECIFICATION READINESS

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, AGM-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
An Effective Specification Development Algorithm Tutorial Outline

2.1 Specification Preparation Readiness
2.2 Specification Preparation Readiness
2.3 Traditional Structured Analysis
2.4 Traditional Structured Analysis
2.5 Unified Modeling Language
2.6 Unified Modeling Language
2.7 Requirements Management
2.8 Requirements Management

Success Is Possible

• The Goal
  – Good specifications
  – On time
  – Affordable

• The Plan
  – A sound beginning - be prepared
  – A clear path to a successful state - always clear what must be done
  – An effective closing - a specification review and approval process
What is a Specification?

A specification contains all of the requirements for a given item.

The Word Requirement, From The Dictionary

- Something wanted or necessary.
- Something essential to the existence or occurrence of something else.
- A necessary characteristic or attribute of some thing (or item).
In Writing a Specification, What Is the Target?

How to Hit the Target of Minimized Completeness

• Every performance requirement traceable to a model from which it was derived
• Every external interface for the item identified and defined in interface requirements in the specification (unless ICD applied)
• Every specialty engineering discipline that has been mapped to the item is included in the specification
• Every environmental influence defined in the appropriate model (system, end item, component) mapped to appropriate specification content.
• Every requirement in the specification traceable to a parent item specification requirement (ideally applies to the system specification relative to user requirements as well).
• Requirements are quantified as appropriate to the statement.
• Requirements are validated (risks understood and mitigated).
Product Requirements Types

- **Hardware**
  - Performance
  - Constraints
    - Interface
    - Specialty Engineering
    - Environmental
- **One view of software requirements**
  - Functional
  - Non-Functional
  - Quality
- **My view of software types**
  - Same as systems and hardware

Requirements Types

All of these requirements must be identified before product and process detailed design work is started and they must be mutually consistent.
Attributes of a Good Requirement

- Achievable (validated)
- Quantified
- Achievable (validated)
- Verifiable/testable
- Unambiguous
- Complete (covers all cases)
- Performance specification
  - Design independent
  - What, not how
- Detail specification
  - Design dependent

Some Good Examples

Frequency coverage. Item frequency coverage shall be 225.0 to 399.9 Megahertz inclusive in tenth Megahertz steps.

Weight. Item weight shall be less than or equal to 240 pounds.

Range. Maximum achievable range shall be greater than or equal to 5,000 nautical miles while recognizing a fuel loading safety margin of 10% or more.

Range. Maximum range shall be greater than or equal to 2,500 nautical miles.

Reliability. Item MTBF shall be greater than or equal to 10,000 hours.
Some Bad Examples

- The screens used in the system shall be designed in a user friendly manner.
- Item weight shall not be greater than 153 pounds.
- Aircraft shall identify their position within 1000 feet of actual along and across track position using Loran C.
- Brakes shall function smoothly and stop the train in a safe distance.
- There shall be no hailstorms in the path of the aircraft.
- On most days, transmitter power output should be 100 watts.
- Go fast.
- Item shall work well and last a long time.
- Any favorites from your past?

Specifications Are Full of Sentences

- THESE SENTENCES SHOULD BE WRITTEN IN THE SIMPLEST POSSIBLE WAY
- THE SUBJECT IS THE ITEM CHARACTERISTIC ABOUT WHICH THE REQUIREMENT IS WRITTEN
- VERB SHALL CLEARLY CALLS FOR COMPLIANCE

<table>
<thead>
<tr>
<th>SUBJECT</th>
<th>VERB</th>
<th>OBJECT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Item memory margin</td>
<td>shall be</td>
<td>greater than 100%</td>
</tr>
</tbody>
</table>

RELATION VALUE
The Verb

- **SHALL**  Mandatory
- **WILL**  Contractor intends to perform
- **SHOULD**  Recommended, Desirable

The Subject

- Writing requirements is easy
- The difficult job is knowing what to write them about - the subject of the sentence
- That is why we model the problem space
Good Examples In Primitive Form

weight < 240 pounds
range > 5,000 nautical miles
MTBF > 10,000 hours

Requirements Analysis Strategies

MODELING

POWER GENERATING SYSTEM

COOLING SYSTEM

VALUE X VALUE Y

CLONING

POWER PLANT

MODELING

ARCHITECTURE SYNTHESIS

ITEM IDENTIFICATION

STRUCTURED DECOMPOSITION

ARCHITECTURE SYNTHESIS

ITEM IDENTIFICATION

ITEM IDENTIFICATION

CUSTOMER Q&A (ELICITATION)

COOLING SYSTEM

POWER PLANT

VALUE X VALUE Y

ITEM IDENTIFICATION

FREESTYLE IS FOR EXPERTS AND FOOLS

FREESTYLE

LIKE ITEM

PARENT ITEM (FLOWDOWN)

COMPONENT STANDARD

LIKE ITEM

PARENT ITEM (FLOWDOWN)

LIKE ITEM

PARENT ITEM (FLOWDOWN)
A Foolproof Search For Subjects

Structured Modeling Tools

Specification Template

DID

Essential Characteristics

Primitive List

Item Specification

Language, Style, Format

System Definition Document

Models

Item Specification

Primitive List

Structured Modeling Tools

Specification Template

DID

General Program Task-Resource Relationships
Work Product Development Suite
Specification Case

Document Progressions
Document Progressions

System Development Process Overview

Covered in this tutorial
Preparatory Steps

A Single Model Will Not Work

System Elements

Software Content

Hardware Content
Hardware and Systems Analysis Models

- Traditional structured analysis
  - Functional analysis
    - Functional flow diagramming
    - Enhanced functional flow diagramming (CORE)
    - Behavioral diagramming (RDD/IPO)
    - IDEF 0 (SADT)
    - Process flow analysis
    - Hierarchical functional analysis
  - Constraints analysis
    - State diagramming
    - SysML

Computer Software Structured Analysis Models

- Process-oriented analysis
  - Flow charting
  - Modern Structured Analysis (Yourdon-Demarco)
  - Modern Structured Analysis (Hatley-Pirbhai)
- Data-oriented analysis
  - Table normalizing
  - IDEF-1X
- Object-oriented analysis
  - Early models
    - Unified Modeling Language (UML)
- DoD architecture framework (DoDAF)
Structured View of a Problem Space

Structured Analysis Methods
Comparison

<table>
<thead>
<tr>
<th>MULTI-FACETED APPROACHES</th>
<th>PRODUCT ENTITY FACET</th>
<th>FUNCTIONAL FACET</th>
<th>BEHAVIORAL FACET</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRADITIONAL STRUCTURED ANALYSIS</td>
<td>PRODUCT ENTITY BLOCK DIAGRAM</td>
<td>FUNCTIONAL FLOW DIAGRAM</td>
<td>SCHEMATIC BLOCK DIAGRAM</td>
</tr>
<tr>
<td>MODERN STRUCTURED ANALYSIS</td>
<td>HIERARCHICAL DIAGRAM</td>
<td>DATA FLOW DIAGRAM</td>
<td>STATE DIAGRAM</td>
</tr>
<tr>
<td>EARLY OBJECT-ORIENTED ANALYSIS</td>
<td>CLASS AND OBJECT DIAGRAM</td>
<td>DATA FLOW DIAGRAM</td>
<td>STATE DIAGRAM</td>
</tr>
<tr>
<td>UML</td>
<td>CLASS/OBJECT, COMPONENT, &amp; DEPLOYMENT DIAGRAMS</td>
<td>USE CASES AND ACTIVITY DIAGRAMS</td>
<td>STATE, SEQUENCE, AND COMMUNICATION DIAGRAMS</td>
</tr>
</tbody>
</table>

○ UNPRECEDENTED ANALYTICAL ENTRY FACET
Model Suggestions for Today

<table>
<thead>
<tr>
<th>SPECIFICATION TYPE</th>
<th>MODEL SUGGESTED</th>
</tr>
</thead>
<tbody>
<tr>
<td>System Specification</td>
<td>Traditional Structured Analysis</td>
</tr>
<tr>
<td>Hardware Performance Specification</td>
<td>Traditional Structured Analysis</td>
</tr>
<tr>
<td>Software Performance Specification, General</td>
<td>Unified Modeling Language (UML)</td>
</tr>
<tr>
<td>Software Performance Specification, Database</td>
<td>IDEF-1X</td>
</tr>
<tr>
<td>DoD IS</td>
<td>DoDAF</td>
</tr>
</tbody>
</table>

But, be prepared to move to the use of SysML coupled with UML and the eventual merge of the two into a more fully integrated common modeling method.

Preparatory Steps
Enabling Documentation

EDG defines the enterprise common process to be applied on all programs, the functional departments, and their process responsibilities which include preparing a department manual covering all common process work allocated to the department.

The system engineering department department manual that defines the way system engineering will be applied on all programs including requirements analysis and specification management.

All functional departments should define templates and data item descriptions for all work products that are prepared as documents. One example of these artifacts is a set of specification data item descriptions that are coordinated with a particular template and a method of acquiring the content through modeling.

Preparatory Steps

[Diagram showing the process flow for preparatory steps in system engineering]
MIL-STD-961E Template

Recommended Template With Map for Traditional Structured Analysis
## Recommended Template With Map for Traditional Structured Analysis

<table>
<thead>
<tr>
<th>PARA</th>
<th>TITLE</th>
<th>MODEL</th>
<th>DEPT</th>
<th>APP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>SCOPE</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>2.</td>
<td>APPLICABLE DOCUMENTS</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.</td>
<td>REQUIREMENTS</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1</td>
<td>Functional and performance requirements</td>
<td>Mission Analysis</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.1</td>
<td>Threat</td>
<td>Threat Analysis</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.3</td>
<td>Required states and modes</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.3.1</td>
<td>Functional analysis</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.3.2</td>
<td>Subordinate entities</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.3.3</td>
<td>Interface relationships</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.3.4</td>
<td>Speciality engineering requirements</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.3.5</td>
<td>Environmental model</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.2</td>
<td>Entity capability requirements</td>
<td>Functional Analysis</td>
<td>2100</td>
<td>A</td>
</tr>
<tr>
<td>3.2.m</td>
<td>Capability m</td>
<td>Functional Analysis</td>
<td>2100</td>
<td>A</td>
</tr>
<tr>
<td>3.3</td>
<td>Interface requirements</td>
<td>N-Square Diagram</td>
<td>2100</td>
<td>D</td>
</tr>
<tr>
<td>3.3.m</td>
<td>External interface requirements</td>
<td>N-Square Diagram</td>
<td>2100</td>
<td>D</td>
</tr>
<tr>
<td>3.3.m.n</td>
<td>Internal interface m, requirement n</td>
<td>N-Square Diagram</td>
<td>2100</td>
<td>D</td>
</tr>
<tr>
<td>3.3.m.n</td>
<td>Internal interface m, requirement n</td>
<td>N-Square Diagram</td>
<td>2100</td>
<td>D</td>
</tr>
<tr>
<td>3.3.m.n</td>
<td>Internal Interface m, requirement n</td>
<td>N-Square Diagram</td>
<td>2100</td>
<td>D</td>
</tr>
</tbody>
</table>

---

## Recommended Template With Map for Traditional Structured Analysis

<table>
<thead>
<tr>
<th>PARA</th>
<th>TITLE</th>
<th>MODEL</th>
<th>DEPT</th>
<th>APP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>SCOPE</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>2.</td>
<td>APPLICABLE DOCUMENTS</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.</td>
<td>REQUIREMENTS</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1</td>
<td>Functional and performance requirements</td>
<td>Mission Analysis</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.1</td>
<td>Threat</td>
<td>Threat Analysis</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.3</td>
<td>Required states and modes</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.3.1</td>
<td>Functional analysis</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.3.2</td>
<td>Subordinate entities</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.3.3</td>
<td>Interface relationships</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.3.4</td>
<td>Speciality engineering requirements</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.3.5</td>
<td>Environmental model</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.2</td>
<td>Entity capability requirements</td>
<td>Functional Analysis</td>
<td>2100</td>
<td>A</td>
</tr>
<tr>
<td>3.2.m</td>
<td>Capability m</td>
<td>Functional Analysis</td>
<td>2100</td>
<td>A</td>
</tr>
<tr>
<td>3.3</td>
<td>Interface requirements</td>
<td>N-Square Diagram</td>
<td>2100</td>
<td>D</td>
</tr>
<tr>
<td>3.3.m</td>
<td>External interface requirements</td>
<td>N-Square Diagram</td>
<td>2100</td>
<td>D</td>
</tr>
<tr>
<td>3.3.m.n</td>
<td>Internal interface m, requirement n</td>
<td>N-Square Diagram</td>
<td>2100</td>
<td>D</td>
</tr>
<tr>
<td>3.3.m.n</td>
<td>Internal interface m, requirement n</td>
<td>N-Square Diagram</td>
<td>2100</td>
<td>D</td>
</tr>
<tr>
<td>3.3.m.n</td>
<td>Internal Interface m, requirement n</td>
<td>N-Square Diagram</td>
<td>2100</td>
<td>D</td>
</tr>
</tbody>
</table>

---

**Modeling Methods Map**

**Functional Department Map**

**Model Artifact Capture Map**
Prepare and Maintain DIDs

- The DID follows the template format
- The recommended DID is focused on a particular modeling approach
- The DID tells how to create a specification using that template with a particular modeling approach

Here is a sample DID for a system specification using TSA.

Preparatory Steps
## Generic Specialty Engineering Scoping Matrix

<table>
<thead>
<tr>
<th>SPECIALTY DISCIPLINE</th>
<th>DEPT</th>
<th>PARA</th>
<th>A</th>
<th>A1</th>
<th>A2</th>
<th>A3</th>
<th>A4</th>
</tr>
</thead>
<tbody>
<tr>
<td>H1</td>
<td>D2164</td>
<td>3.4.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H2</td>
<td>D2164</td>
<td>3.4.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H3</td>
<td>D2164</td>
<td>3.4.3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H4</td>
<td>D2164</td>
<td>3.4.4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H5</td>
<td>D2311</td>
<td>3.4.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H6</td>
<td>D2311</td>
<td>3.4.51</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H7</td>
<td>D2113</td>
<td>3.4.52</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H8</td>
<td>D2311</td>
<td>3.4.53</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H9</td>
<td>D2311</td>
<td>3.4.54</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HA</td>
<td>D2313</td>
<td>3.4.55</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HB</td>
<td>D2313</td>
<td>3.4.56</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HC</td>
<td>D2165</td>
<td>3.4.6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HD</td>
<td>D2160</td>
<td>3.4.7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HE</td>
<td>D2136</td>
<td>3.4.9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HF</td>
<td>D2136</td>
<td>3.4.10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HG</td>
<td>D2136</td>
<td>3.4.11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HI</td>
<td>D2168</td>
<td>3.4.12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HJ</td>
<td>D5100</td>
<td>3.4.13</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HK</td>
<td>D5100</td>
<td>3.4.14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HL</td>
<td>D2167</td>
<td>3.4.142</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SPECIALTY DISCIPLINE</th>
<th>DEPT</th>
<th>PARA</th>
<th>A</th>
<th>A1</th>
<th>A2</th>
<th>A3</th>
<th>A4</th>
</tr>
</thead>
<tbody>
<tr>
<td>HM</td>
<td>D5100</td>
<td>3.4.14.3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HN</td>
<td>D2113</td>
<td>3.4.14.4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HO</td>
<td>D2113</td>
<td>3.4.14.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HP</td>
<td>D2124</td>
<td>3.4.14.6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HQ</td>
<td>D2124</td>
<td>3.4.14.7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HR</td>
<td></td>
<td>3.4.14.8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HS</td>
<td>D2123</td>
<td>3.4.14.9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HT</td>
<td>D2144</td>
<td>3.4.14.10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HU</td>
<td>D2143</td>
<td>3.4.14.11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HV</td>
<td></td>
<td>3.4.14.12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HW</td>
<td></td>
<td>3.4.14.13</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HX</td>
<td>D2167</td>
<td>3.4.14.14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Proposal Team Specification Actions

Environmental Requirements

- **System**
  - Identify spaces within which the system will have to function
  - Select standards covering those spaces
  - For each standard, select parameters that apply
  - Tailor the range of selected parameters

- **End item**
  - Build three dimensional model of end items, physical processes, and process environments
  - Extract item environments

- **Component**
  - Zone end item into spaces of common environmental characteristics
  - Map components to zones
  - Components inherit zone environmental requirements
Specialty Engineering Scoping Matrix
Applied to Program

Configuration Control the Models
On To Program Work
AN EFFECTIVE SPECIFICATION DEVELOPMENT ALGORITHM TUTORIAL

TRADITIONAL STRUCTURED ANALYSIS

The Big Bang Theory of System Development
THE TRADITIONAL APPROACH

EVERYTHING FLOWS FROM ONE IDEA,
THE CUSTOMER NEED
IT IS THE ULTIMATE REQUIREMENT,
THE ULTIMATE FUNCTION

BA-BA-BA-BANG
Two Top-Level Views of a System

The Beginning Of Functional Decomposition
Traditional Structured Analysis Model
Overview

The Ultimate Function and Its First Expansion

Alternative functional analysis techniques
- Enhanced functional flow block diagramming (CORE)
- Behavioral diagramming (RDD)
- IDEF-0
Example of a Life Cycle Model

Use System Expansion Example
Space Transport System
Continued Function Decomposition

An orderly exposure of needed functionality moving from the known to the unknown, from simple to the complex, from the top to the bottom.

Performance Requirements Analysis and the RAS

<table>
<thead>
<tr>
<th>FUNCTION</th>
<th>PRODUCT ENTITY</th>
<th>REQUIREMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>MID</td>
<td>NAME</td>
<td>PID</td>
</tr>
<tr>
<td>F123</td>
<td>Provide Thrust</td>
<td>A12 Engine</td>
</tr>
</tbody>
</table>

Exposing what the system must do and how well it must do it encouraging identification of all essential characteristics and avoidance of unnecessary characteristics.
The Function-Product Entity Plane

Functional Analysis Alternatives

- **IDEF 0**
  - A variation on SADT
- **Behavioral Diagramming**
  - From Ascent Logic’s RDD
  - Based on IPO
- **Enhanced Functional Flow Block Diagramming**
  - Employed in Vitech’s CORE
- **Hierarchical Functional Block Diagramming**
Requirements Capture Using the RAS-Complete Format

Here Is What We Want

<table>
<thead>
<tr>
<th>Module ID</th>
<th>Requirement Entry</th>
<th>Product Entry</th>
<th>Document Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>Name</td>
<td>RD</td>
<td>Text</td>
</tr>
<tr>
<td>Performance Requirements</td>
<td>Frame Requirements</td>
<td>Interface Requirements</td>
<td>Specialty Engineering Requirements</td>
</tr>
</tbody>
</table>

This Is How To Get It

Product Entity Structure

Use a common structure that includes hardware and software.

- TSA
- UML
- SYSTEM A
  - A1
  - A2
  - A3
  - A11
  - A12
  - A13
  - A14
  - A15
Systems Consist of Things and Relationships

Organizing For Interface Development

- Decompose needed functionality and allocate to product entities
- Map product entities to responsible development organizations
  - Create cross-functional integrated product and process teams (IPPT)
  - Assign principal engineers for lowest tier responsibilities on teams (everything has someone responsible)
- Establish clear rules for interface development responsibility
  - Identify needed interfaces as a function of how functionality was allocated to entities
  - Analyze product entity pair relationships using n-square diagrams
  - Partition interface into subsets as a function of product entity principal views
  - Assign interface responsibility to product entity principal engineers as a function of a receiving terminal rule (if you need an interface you must come forward)
  - System engineering manage the aggregate external and inter-team interface sets applying a lowest common team integration concept
- Minimize external (cross-organizational) interface at all levels, iterating product entity structure and/or development organization responsibilities to do so, if necessary, then apply system engineering integration resources to that which remains
Two Interface Definition Models

SCHEMATIC BLOCK DIAGRAMMING

- Lines define interfaces
- Blocks are objects only from the product entity structure

N-SQUARE DIAGRAMMING

- Marked intersections define interfaces
- Diagonal blocks are objects only from product entity block diagram
- Apparent ambiguity reflects directionality

Interface Requirements Derivation

Geometrical View
Development Often Fails at the Cross-organizational Interfaces

Interface Integration Focus
The Fundamental Problems in Interface Work

There is a one-to-one correspondence between teams and components. There is a one-to-two correspondence between teams and interfaces.

We tend to focus inwardly.
The Fundamental Problems in Interface Work

We are dependent on the worst interface on planet Earth in the development of interfaces.

Benefits Of Product Team Organization

WBS, SOW, IMP, IMS
Budget and schedule Specifications
Inter-team Communication Requirement
Cross-Organizational Interface Requirement

Interface Control Working Teams
Component Teams
Specialty Engineering Identification of Constraints

SPECIALTY ENGINEERING CONSTRAINTS MATRIX

SPECIALTY ENGINEERING REQUIREMENTS FLOW INTO THE INDICATED SPECIFICATIONS VIA THE RAS IMPLEMENTED IN A DATABASE

Specialty Engineering Plane Added
Environment Subsets

Environmental Requirements

- **System**
  - Identify spaces within which the system will have to function
  - Select standards covering those spaces
  - For each standard, select parameters that apply
  - Tailor the range of selected parameters

- **End item**
  - Build three dimensional model of end items, physical processes, and process environments
  - Extract item environmental requirements

- **Component**
  - Zone end item into spaces of common environmental characteristics
  - Map components to zones
  - Components inherit zone environmental requirements
### Performance Requirements

<table>
<thead>
<tr>
<th>Requirement Entry Name</th>
<th>Requirement Entry Number</th>
<th>Requirement</th>
<th>Product Entity Name</th>
<th>Product Entity Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

### Specialty Engineering

<table>
<thead>
<tr>
<th>Requirement Entry Name</th>
<th>Requirement Entry Number</th>
<th>Requirement</th>
<th>Product Entity Name</th>
<th>Product Entity Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

### Interface

<table>
<thead>
<tr>
<th>Requirement Entry Name</th>
<th>Requirement Entry Number</th>
<th>Requirement</th>
<th>Product Entity Name</th>
<th>Product Entity Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

### Environmental

<table>
<thead>
<tr>
<th>Requirement Entry Name</th>
<th>Requirement Entry Number</th>
<th>Requirement</th>
<th>Product Entity Name</th>
<th>Product Entity Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

---

**Environmental Planes Added**

**RAS Complete In Tabular Form**

---

**NDIA**

- National Defense Industrial Association

**JOG System Engineering, Inc.**
Traditional Structured Analysis

Process View

Template/DID

System Definition Document

NOTE
Specification template paragraph numbers in red.

Save the Models!
UNIFIED MODELING LANGUAGE

Agenda

• The Dead Sea scrolls of software development
• UML modeling artifacts
• UML modeling approach
• Integration
• Requirements and modeling documentation
Software Dead Sea Scrolls

- Flow chartings
  - Functionality examined
  - Data in the back seat
- SADT and IPO
  - Two axis models
- Modern structured analysis and HP
  - Functionality and data examined
- Early OOA
  - Search for objects and their behavior
  - Anti Sullivan

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 tutorial - form follows function.

- Dynamically model the problem space from the top-down
- Identify responsible classifiers
- Package classifiers from the top-down

UML can support either direction.

Note: A classifier is a general term for a software product entity represented by a node, component, or class in UML.
The Software Development Process

- Identify a product entity that will be developed as computer software.
- Dynamically analyze the entity.
  - Use cases
  - Sequence diagram
  - Communication diagram
  - activity diagram
  - State diagram
- In the sequence, communication, and activity diagramming analysis you will have to identify lower tier 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.

Consolidated System Product Entity Structure
# Suggested SRS Structure

## 3 REQUIREMENTS

### 3.1 Required states and modes

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.2</td>
<td>Software entity capability requirements</td>
</tr>
<tr>
<td>3.2.m</td>
<td>Software entity capability m</td>
</tr>
<tr>
<td>3.2.m.n</td>
<td>Software entity capability m, requirement n</td>
</tr>
<tr>
<td>3.3</td>
<td>Software entity interface requirements</td>
</tr>
<tr>
<td>3.3.1</td>
<td>Software entity external interface requirements</td>
</tr>
<tr>
<td>3.3.1.m</td>
<td>Specific external interface m</td>
</tr>
<tr>
<td>3.3.1.m.n</td>
<td>External interface m, requirement n</td>
</tr>
<tr>
<td>3.3.2</td>
<td>Software entity internal interface requirements</td>
</tr>
<tr>
<td>3.3.2.m</td>
<td>Specific internal interface m</td>
</tr>
<tr>
<td>3.3.2.m.n</td>
<td>Internal interface m, requirement n</td>
</tr>
<tr>
<td>3.3.3</td>
<td>Software entity internal data requirements</td>
</tr>
<tr>
<td>3.3.3.m.n</td>
<td>Specific software entity internal data requirement n</td>
</tr>
</tbody>
</table>

### 3.4 Specialty engineering requirements

### 3.5 Software entity environmental requirements

### 3.6 Precedence and criticality requirements

## The Diagrams of UML

- For modeling dynamic aspects of the system
  - Use case diagram
  - Sequence diagram
  - Timing diagram
  - Communication diagram (renamed in 2)
  - State diagram
  - Activity 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)

(2) = added in UML 2.0
The Dynamic Models

Sequence Diagram UX-11321

Activity Diagram UX11323

State Diagram UX11324

Interaction Diagrams

Communication Diagram UX11322

Sequence Diagram UX-11321
Emphasizes the time ordering of messages

Actor

messageOne()

messageTwo()

messageThree()

messageFour()

messageFive()

Classifier AX1

Classifier AX2

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

• A message is the specification of a communication among objects on a class or object diagram or between the objects represented by life lines on the sequence diagram or blocks of a communication diagram.

• When a message is passed from one object to another some action usually results on its receipt.

• The action may result in a change of state in the object on the arrow head.

• State related requirements in terms related to the target object.

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
A Simple Example

Communication Diagram UX11322
Emphasizes structural relationships

Semantically identical to the sequence diagram.
Activity Diagram UX11323

State Diagram UX11324
The Static Entities in UML

- **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 node or collection of collection of nodes.

- **Node**
  - Appears on a deployment diagram that exists at run time and 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 classes.

- **Class**
  - A description of a set of objects that share the same attributes, operations, relationships, and semantics.

- **Object**
  - An instance of a class.

---

UML Structural Artifacts in a Product
Entity Structure
Deployment and Component Diagrams

Classes and Objects

A class is a description of a set of objects that share the same attributes, operations, relationships, and semantics. An object is an instance of a class. Graphically a class is rendered as a rectangle.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>The name is a noun or noun phrase</td>
</tr>
<tr>
<td>attributeOne, attributeTwo, attributeThree, attributeFour</td>
<td>An attribute is a named property of a class that describes a range of values that instances of the property may hold. An attribute represents some property of the thing you are modeling that is shared by all objects of that class.</td>
</tr>
<tr>
<td>operationOne(), operationTwo(), operationThree()</td>
<td>An operation is the implementation of a service that can be requested from any object of a class to effect behavior. An operation is an abstraction of something you can do to an object that is shared by all objects of that class</td>
</tr>
</tbody>
</table>
Class Responsibilities

A responsibility is a contract or an obligation of a class. You may find it useful to begin the analysis of classes this way translating these into attributes and operations that best fulfill the class's responsibility as the model is refined. The responsibility is noted in an added compartment in which descriptive free form text is entered.

Structural Relationships

These Have Nothing To Do With Messages

NameOne is the base class. NameTwo and NameThree are leaf classes in a generalization. NameOne depending on NameFour for information and services.

An association is a structural relationship.
Structural Relationships
Association Adornments

- Association Name
  NameOne
  x.y
  Association name
  role
  name direction
  NameTwo
  role

- Association Role
  The face that the class at the far end of an association presents to the class at near end of the association. Role names called end names.

- Association Multiplicity
  Tells how many objects may be connected across an association instance. Given by a range of numbers.

- Association Aggregation
  Expresses a whole-part relationship between to associated classes.

A Flexible Dynamic Modeling Overview

Specifications
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 use case.
- For each scenario build a sequence diagram and in the process identify next lower tier classifiers and messages between the actors and lower tier classifiers.
- Apply communication, activity, and state diagrams as needed.
- Derive requirements from dynamic modeling artifacts.

Hierarchical Modeling Relationships

![Diagram showing hierarchical modeling relationships]
Context Diagram

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

Identify one or more use cases for each terminator.

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

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

Use Case Fundamentals

- A use case is a more expressive context diagram common 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

- **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 UX-11

Actors derive benefits from the system.

Use Case UX-11

Extended Use Case UX-111

Extended Use Case UX-112

Extended Use Case UX-113

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

Possible Multiple Scenarios

Extended Use Case UX-113

Scenario UX-1131

Scenario UX-1132

Scenario UX-1133

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

Textual scenario descriptions
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.

Examine Each Scenario Dynamically

- Activity, sequence, and communication diagrams require identification of lower tier entities leading to additional of entities on the consolidated product entity diagram.
- State diagrams may also be useful in identifying essential characteristics appropriate for the entity being analyzed.
- Requirements flow out of the dynamic analysis and into the specification for the entity being analyzed.
AN EFFECTIVE SPECIFICATION DEVELOPMENT ALGORITHM TUTORIAL

REQUIREMENTS MANAGEMENT

Agenda

- Process summary
- Organizing and specification responsibility
- Requirements risk management
- Traceability
- Tools
- Review, approval, release, and distribution
- A peek into the future
Life Cycle Model

Suggested Enterprise Structure
Benefits of Product-Oriented Team Structure

The System Product Entity Structure and Teaming
Risk Defined

- The danger that injury, damage, or loss will occur
- Somebody or something likely to cause injury, damage, or loss
- The probability, amount, or type of possible loss incurred by an insurer
- The possibility of loss in an investment or speculation
- The statistical chance of danger from something, especially from the failure of an engineered system

Risk Measurement Parameters

### Probability of Occurrence

<table>
<thead>
<tr>
<th>CAT</th>
<th>TITLE</th>
<th>P(O)</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Nearly Certain</td>
<td>0.95-1.00</td>
<td>Will occur at least once during program</td>
</tr>
<tr>
<td>4</td>
<td>Probable</td>
<td>0.75-0.95</td>
<td>Will probably occur once during program</td>
</tr>
<tr>
<td>3</td>
<td>Possible</td>
<td>0.50-0.75</td>
<td>May occur during program</td>
</tr>
<tr>
<td>2</td>
<td>Unlikely</td>
<td>0.25-0.49</td>
<td>Will probably not occur during program</td>
</tr>
<tr>
<td>1</td>
<td>Nearly Impossible</td>
<td>0.00-0.24</td>
<td>Will not occur during program</td>
</tr>
</tbody>
</table>

### Seriousness of Effect

<table>
<thead>
<tr>
<th>CAT</th>
<th>TITLE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Catastrophic</td>
<td>Program in jeopardy of termination</td>
</tr>
<tr>
<td>4</td>
<td>Serious</td>
<td>Serious damage to program</td>
</tr>
<tr>
<td>3</td>
<td>Moderate</td>
<td>Problems cause program focus difficulties</td>
</tr>
<tr>
<td>2</td>
<td>Minor</td>
<td>Problems that can be easily be overcome</td>
</tr>
<tr>
<td>1</td>
<td>Null</td>
<td>No problem</td>
</tr>
</tbody>
</table>
### Risk Metric Values

<table>
<thead>
<tr>
<th>SERIOUSNESS OF THE EFFECT</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>25</td>
<td>20</td>
<td>15</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>20</td>
<td>16</td>
<td>12</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>15</td>
<td>12</td>
<td>9</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>8</td>
<td>6</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

Legend:
- Dark gray: High Risk
- Middle gray: Medium Risk
- Light gray: Low Risk

### Aggregate Program Risk Index

[Graph depicting the aggregate program risk index over time with peak risk points]
Traceability Forms

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

Traceability Integration

• Lateral traceability within TSA and UML independently should be no problem with all requirements derived from models
• Vertical interface traceability is decomposition driven within TSA and UML as well as across the HW-SW gap
• Environmental requirements are significantly different between HW and SW so traceability is not a significant issue between them
• Performance requirements across the HW-SW gap offer a vertical traceability challenge
The System Product Entity Structure

This is the kind of relationship of interest

Level at which a subordinate software entity is identified

Hardware entity
Software entity
System

FUNCTION U
FUNCTION V
FUNCTION W
FUNCTION X

RAS
PRODUCT ENTITY
PRODUCT ENTITY
PRODUCT ENTITY
PRODUCT ENTITY

SOFTWARE ENTITY
IDENTIFIED THROUGH TSA

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.

Entity AX2
Context Diagram Terminator UX21
Use Case UX21
Extended Use Case UX2111
Scenario UX21111

Sequence Diagram UX211111
Communication Diagram UX211112
Activity Diagram UX211113
State Diagram UX211114
### Traceability Evaluation Matrix

**Requirements Derived From UML Modeling**

<table>
<thead>
<tr>
<th></th>
<th>R%1</th>
<th>R%2</th>
<th>R%3</th>
<th>R%4</th>
<th>R%5</th>
<th>R%6</th>
<th>R%7</th>
<th>R%8</th>
<th>R%9</th>
<th>R%10</th>
<th>R%11</th>
<th>R%12</th>
</tr>
</thead>
<tbody>
<tr>
<td>R%1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R%2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R%3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R%4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R%5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R%6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R%7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R%8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R%9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R%10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**ACTUAL RID EXAMPLE**

- R%1: RUTZ7H
- R%2: R9ER6
- R%3: R937YF
- R%4: RJ8EG
- R%5: RJT6T
- R%6: RHGT5T
- R%7: R5D7W
- R%8: RJ98S7
- R%9: RL34DF
- R%10: R456HD

Alternatively, one could rely upon experienced inspection without the organizing influence of the matrix.
Today’s Tools

Tools Integration
Tomorrow’s Tools

• Front end modeling tools
  – Use case modeling
  – Function/activity modeling
  – State modeling (behavioral modeling)
  – Sequence/timeline modeling
  – Product entity and interface modeling
  – Specialty engineering database linkage
  – Environmental coverage

• Connection of modeling to management database

• Big dumb database
  – Requirements capture
  – Traceability
  – Value management
  – Specification publishing

Tools Integration
Configuration Management of Requirements Documentation

- System Engineering Managed Content
- Configuration Managed Content

Requirements Analysis → Database Content → Publish and Release → Library

- Computer Projection Review

Portion of database corresponding to released specifications and library content under formal configuration control.
Utility Of Computer Projection

- ON-LINE NETWORK CAPABILITY
- PUT THE PROJECTION CAPABILITY IN THE WORK AREA
- APPLY REAL-TIME CONCURRENT DEVELOPMENT (IPD)
- FORM AND REFORM BETWEEN MEETING AND INDIVIDUAL WORK QUICKLY

Specification Review and Approval Process
Evaluate for Template Faithfulness

- Compare specification cover data with template (standard)
- Compare specification paragraphing structure with template
- Compare specification style with template style guide

Individual Requirement Quality

- Spot check specification requirements for requirements quality checklist compliance
- Spot check specification for requirements quantification where appropriate
Section 2 Traceability

- All documents listed in Section 2 called somewhere in the specification
- All documents tailored, if necessary, to limit coverage to the application
- All documents called in the requirements listed in Section 2
- Spot check for excessively tailored standards which could be quoted instead of being called
- Ensure documents called are current and accepted authorities for the application

Completeness and Avoidance of Unnecessary Content

- Ask principal engineer how content was derived
  - If ad hoc, there should be concern
  - If through structured analysis, spot check how a few requirements were derived (ask to see the supporting modeling data)
- Ensure all requirements traceable to parent requirements
Residual Risk Evaluation

- All TBD/TBR are closed or, if not, are being carried as program or team risks
- An approved concept exists

Movement To Universal Method
UML and Functional Analysis

Unified Modeling Language (UML)

<table>
<thead>
<tr>
<th>STATIC DIAGRAMS</th>
<th>DYNAMIC DIAGRAMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEPLOYMENT DIAGRAM</td>
<td>COMPONENT DIAGRAM</td>
</tr>
<tr>
<td>OBJECT &amp; CLASS DIAGRAMS</td>
<td>STATE CHART</td>
</tr>
<tr>
<td>INTERACTION DIAGRAMS</td>
<td>COMMUNICATION DIAGRAM</td>
</tr>
<tr>
<td>SEQUENCE DIAGRAM</td>
<td>USE CASE DIAGRAM</td>
</tr>
<tr>
<td>ACTIVITY DIAGRAM</td>
<td></td>
</tr>
<tr>
<td>ARCHITECTURE BLOCK DIAGRAM</td>
<td>STATE DIAGRAM</td>
</tr>
<tr>
<td>SCHEMATIC BLOCK DIAGRAM</td>
<td>TIMELINE DIAGRAM</td>
</tr>
<tr>
<td>FUNCTIONAL FLOW DIAGRAM</td>
<td></td>
</tr>
<tr>
<td>PHYSICAL FACET</td>
<td>BEHAVIORAL FACET</td>
</tr>
<tr>
<td>FUNCTIONAL FACET</td>
<td></td>
</tr>
</tbody>
</table>

Traditional Structured Analysis
A Subset of UML?

Modeling Changes In the Near Term

Component Diagram
Deployment Diagram
Communication Diagram
Interaction Overview Diagram
Package Diagram

UNIVERSAL MODEL
OF THE FUTURE

UML

SysML

Requirements Diagram
Parametric Diagram
Assembly Diagram

PUSH THESE COMPONENTS TOGETHER MORE TIGHTLY

REPLACING TSA

SysML DERIVED FROM UML
System Modeling Evolution Timeline

Over the Hill and Through the Woods to Utopia
Review and Summary

The target is completeness and avoidance of unnecessary content

Use models to identify essential characteristics

Do the analysis

Write requirements in primitive form based on essential characteristics identified through modeling.

Translate into full sentences and insert into paragraph structure.
GRAND SYSTEMS
DEVELOPMENT TRAINING
PROGRAM

VERSION 10.1

The union of system engineering, domain engineering, functional management, and program management for the greater good of the enterprise and customer base.

VOLUME 2R
AN EFFECTIVE SPECIFICATION DEVELOPMENT ALGORITH

Presented By
Jeffrey O. Grady

JOG - SYSTEM ENGINEERING, Inc.

6015 Charae Street
San Diego, California 92122
(858) 458-0121
(858) 456-0867 Fax
jgrady@ucsd.edu or jeff@jogse.com
http://www.jogse.com

Copyright 2006

No part of this manual may be scanned or reproduced in any form without permission in writing from the author.
<table>
<thead>
<tr>
<th>PARAGRAPH</th>
<th>TITLE</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Specification Templates and DIDs</td>
<td>2</td>
</tr>
<tr>
<td>1.1</td>
<td>Enterprise Engineering Work</td>
<td>2</td>
</tr>
<tr>
<td>1.2</td>
<td>System Engineering Generic Work</td>
<td>3</td>
</tr>
<tr>
<td>1.3</td>
<td>Proposal Work That Prepares for Program Execution</td>
<td>4</td>
</tr>
<tr>
<td>1.4</td>
<td>Work Subsequent to Contract Award</td>
<td>6</td>
</tr>
<tr>
<td>1.5</td>
<td>The Preferred Templates</td>
<td>7</td>
</tr>
<tr>
<td>1.6</td>
<td>Modeling Work Product Capture Document</td>
<td>9</td>
</tr>
<tr>
<td>2</td>
<td>Structured Analysis</td>
<td>9</td>
</tr>
<tr>
<td>2.1</td>
<td>Traditional Structured Analysis</td>
<td>9</td>
</tr>
<tr>
<td>2.1.1</td>
<td>A System Defined</td>
<td>11</td>
</tr>
<tr>
<td>2.1.2</td>
<td>The System Environment</td>
<td>11</td>
</tr>
<tr>
<td>2.1.3</td>
<td>System Functionality</td>
<td>12</td>
</tr>
<tr>
<td>2.1.4</td>
<td>Performance Requirements Derivation and Allocation</td>
<td>15</td>
</tr>
<tr>
<td>2.1.5</td>
<td>Product Entity Structure</td>
<td>15</td>
</tr>
<tr>
<td>2.1.6</td>
<td>Allocation Pacing Alternatives</td>
<td>17</td>
</tr>
<tr>
<td>2.1.7</td>
<td>System Relations</td>
<td>18</td>
</tr>
<tr>
<td>2.1.8</td>
<td>Environmental Relation Algorithm</td>
<td>20</td>
</tr>
<tr>
<td>2.1.8.1</td>
<td>System Environmental Relations</td>
<td>20</td>
</tr>
<tr>
<td>2.1.8.2</td>
<td>End Item Service Use Profile</td>
<td>20</td>
</tr>
<tr>
<td>2.1.8.3</td>
<td>Component Environmental Relations</td>
<td>21</td>
</tr>
<tr>
<td>2.1.8.4</td>
<td>Environmental Impact</td>
<td>22</td>
</tr>
<tr>
<td>2.1.9</td>
<td>Specialty Engineering and RAS Complete</td>
<td>22</td>
</tr>
<tr>
<td>2.1.10</td>
<td>RAS-Complete in Table Form</td>
<td>24</td>
</tr>
<tr>
<td>2.1.11</td>
<td>Traditional Structured Analysis Summary</td>
<td>25</td>
</tr>
<tr>
<td>2.1.12</td>
<td>SDD Content and Format</td>
<td>26</td>
</tr>
<tr>
<td>2.1.12.1</td>
<td>Document Main Body</td>
<td>27</td>
</tr>
<tr>
<td>2.1.12.2</td>
<td>Appendix A, Functional Analysis</td>
<td>27</td>
</tr>
<tr>
<td>2.1.12.3</td>
<td>Appendix B, System Environment Analysis</td>
<td>27</td>
</tr>
<tr>
<td>2.1.12.4</td>
<td>Appendix C, System Architecture Analysis</td>
<td>28</td>
</tr>
<tr>
<td>2.1.12.5</td>
<td>Appendix D, System Interface Analysis</td>
<td>28</td>
</tr>
<tr>
<td>2.1.12.6</td>
<td>Appendix E, Specialty Engineering Definition Analysis</td>
<td>28</td>
</tr>
<tr>
<td>2.1.12.7</td>
<td>Appendix F, System Process Analysis</td>
<td>28</td>
</tr>
<tr>
<td>2.1.12.8</td>
<td>Appendix G, Requirements Analysis Sheet</td>
<td>28</td>
</tr>
<tr>
<td>2.1.13</td>
<td>Team Activity During Requirements Work</td>
<td>29</td>
</tr>
<tr>
<td>2.2</td>
<td>UML</td>
<td>30</td>
</tr>
<tr>
<td>2.2.1</td>
<td>Entry Analysis and Overview</td>
<td>30</td>
</tr>
<tr>
<td>2.2.2</td>
<td>The Connection Between Modeling Artifacts, Specification</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td>Content, and Product Entities</td>
<td></td>
</tr>
<tr>
<td>2.2.3</td>
<td>Dynamic Modeling Artifacts Explained</td>
<td>35</td>
</tr>
<tr>
<td>2.2.3.1</td>
<td>Sequence Diagram</td>
<td>35</td>
</tr>
<tr>
<td>2.2.3.2</td>
<td>Communication Diagram</td>
<td>37</td>
</tr>
<tr>
<td>2.2.3.3</td>
<td>Activity Diagram</td>
<td>37</td>
</tr>
</tbody>
</table>
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>PARAGRAPH</th>
<th>TITLE</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.2.3.4</td>
<td>State Diagram</td>
<td>38</td>
</tr>
<tr>
<td>2.2.4</td>
<td>Structural Analysis</td>
<td>39</td>
</tr>
<tr>
<td>2.2.4.1</td>
<td>The Class</td>
<td>40</td>
</tr>
<tr>
<td>2.2.4.2</td>
<td>Class Relationships</td>
<td>41</td>
</tr>
<tr>
<td>2.2.4.3</td>
<td>Messages</td>
<td>42</td>
</tr>
<tr>
<td>2.2.5</td>
<td>Related Analyses</td>
<td>42</td>
</tr>
<tr>
<td>2.2.5.1</td>
<td>Specialty Engineering</td>
<td>42</td>
</tr>
<tr>
<td>2.2.5.2</td>
<td>Environmental Requirements</td>
<td>42</td>
</tr>
<tr>
<td>2.2.6</td>
<td>Specification Structure</td>
<td>42</td>
</tr>
<tr>
<td>2.2.7</td>
<td>Software Requirements Close-out</td>
<td>44</td>
</tr>
<tr>
<td>2.3</td>
<td>Opening the Analysis With DoDAF</td>
<td>45</td>
</tr>
<tr>
<td>2.4</td>
<td>Integrated Modeling</td>
<td>47</td>
</tr>
<tr>
<td>3.1</td>
<td>Requirements Management</td>
<td>51</td>
</tr>
<tr>
<td>3.2</td>
<td>Summary of Team Activity During Requirements Work</td>
<td>51</td>
</tr>
<tr>
<td>3.3</td>
<td>Requirements Tools Base</td>
<td>52</td>
</tr>
<tr>
<td>3.4</td>
<td>Recommended Specification Responsibility Pattern</td>
<td>53</td>
</tr>
<tr>
<td>3.4.1</td>
<td>Requirements Risk Management</td>
<td>54</td>
</tr>
<tr>
<td>3.4.2</td>
<td>Requirements Validation</td>
<td>54</td>
</tr>
<tr>
<td>3.4.3</td>
<td>Margins and Budgets</td>
<td>55</td>
</tr>
<tr>
<td>3.5</td>
<td>Risk Tracking</td>
<td>55</td>
</tr>
<tr>
<td>3.6</td>
<td>Verification Requirements</td>
<td>58</td>
</tr>
<tr>
<td>3.6</td>
<td>Specification Review and Approval</td>
<td>59</td>
</tr>
<tr>
<td>A</td>
<td>APPENDIX A, PRESENTATION MATERIALS</td>
<td>A-i</td>
</tr>
<tr>
<td>B</td>
<td>APPENDIX B, SPECIFICATION DATA ITEM DESCRIPTIONS</td>
<td>B-i</td>
</tr>
<tr>
<td></td>
<td>JOGSE System Specification Data Item Description</td>
<td>B-1-1</td>
</tr>
<tr>
<td></td>
<td>JOGSE Hardware Item Performance Specification Data Item Description</td>
<td>B-2-1</td>
</tr>
<tr>
<td></td>
<td>JOGSE Software Requirements Specification Data Item Description</td>
<td>B-3-1</td>
</tr>
</tbody>
</table>

**NOTE**

Exhibit B available from the lecturer by sending an email to jgady@ucsd.edu and requesting it.
# LIST OF ILLUSTRATIONS

<table>
<thead>
<tr>
<th>FIGURE</th>
<th>TITLE</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Overall Process</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>Preparatory Steps</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>Proposal Team Work</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>Program Work</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>System and Hardware Specification Template</td>
<td>8</td>
</tr>
<tr>
<td>6</td>
<td>Overview of the Traditional Structured Analysis Process</td>
<td>10</td>
</tr>
<tr>
<td>7</td>
<td>Ultimate System Diagram</td>
<td>11</td>
</tr>
<tr>
<td>8</td>
<td>System Environment</td>
<td>12</td>
</tr>
<tr>
<td>9</td>
<td>System Context Diagram</td>
<td>12</td>
</tr>
<tr>
<td>10</td>
<td>Function Sequence</td>
<td>13</td>
</tr>
<tr>
<td>11</td>
<td>Function Decomposition</td>
<td>14</td>
</tr>
<tr>
<td>12</td>
<td>System Life Cycle</td>
<td>14</td>
</tr>
<tr>
<td>13</td>
<td>Traditional Requirements Analysis Sheet</td>
<td>15</td>
</tr>
<tr>
<td>14</td>
<td>Product Entity Structure</td>
<td>16</td>
</tr>
<tr>
<td>15</td>
<td>Juxtaposition of RAS and N-square Diagrams</td>
<td>18</td>
</tr>
<tr>
<td>16</td>
<td>A Geometric View of the RAS Complete</td>
<td>23</td>
</tr>
<tr>
<td>17</td>
<td>RAS-Complete in Tabular Form</td>
<td>25</td>
</tr>
<tr>
<td>18</td>
<td>The System Relationship</td>
<td>26</td>
</tr>
<tr>
<td>19</td>
<td>SDD Structure</td>
<td>27</td>
</tr>
<tr>
<td>20</td>
<td>Context Diagram</td>
<td>30</td>
</tr>
<tr>
<td>21</td>
<td>Unified Modeling Language Overview</td>
<td>31</td>
</tr>
<tr>
<td>22</td>
<td>Hierarchical Relationship Between UML Dynamic Modeling Artifacts</td>
<td>33</td>
</tr>
<tr>
<td>23</td>
<td>Sequence Diagram Example</td>
<td>36</td>
</tr>
<tr>
<td>24</td>
<td>Communication Diagram Example</td>
<td>37</td>
</tr>
<tr>
<td>25</td>
<td>Activity Diagram Example</td>
<td>38</td>
</tr>
<tr>
<td>26</td>
<td>State Diagram Example</td>
<td>39</td>
</tr>
<tr>
<td>27</td>
<td>The UML Classifiers</td>
<td>40</td>
</tr>
<tr>
<td>28</td>
<td>Structural Relationships</td>
<td>41</td>
</tr>
<tr>
<td>29</td>
<td>Association Adornments</td>
<td>41</td>
</tr>
<tr>
<td>30</td>
<td>Software requirements specification template</td>
<td>43</td>
</tr>
<tr>
<td>31</td>
<td>Evolving Product Entity Structure</td>
<td>45</td>
</tr>
<tr>
<td>32</td>
<td>DoDAF Development Sequence</td>
<td>46</td>
</tr>
<tr>
<td>33</td>
<td>Requirements Traceability Across the Gap</td>
<td>49</td>
</tr>
<tr>
<td>34</td>
<td>Modeling Over the Years</td>
<td>50</td>
</tr>
<tr>
<td>35</td>
<td>The Approaching Merge</td>
<td>51</td>
</tr>
<tr>
<td>36</td>
<td>Tools Environment</td>
<td>53</td>
</tr>
<tr>
<td>37</td>
<td>Risk Matrix</td>
<td>56</td>
</tr>
<tr>
<td>38</td>
<td>Program Risk Tracking Chart</td>
<td>56</td>
</tr>
<tr>
<td>39</td>
<td>Verification Traceability</td>
<td>57</td>
</tr>
<tr>
<td>40</td>
<td>Specification Review and Approval</td>
<td>59</td>
</tr>
</tbody>
</table>
# LIST OF TABLES

<table>
<thead>
<tr>
<th>TABLE</th>
<th>TITLE</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Independent and Combined SDD Appendices</td>
<td>44</td>
</tr>
<tr>
<td>2</td>
<td>Risk Probability of Occurrence Criteria</td>
<td>56</td>
</tr>
<tr>
<td>3</td>
<td>Risk Effects Criteria</td>
<td>56</td>
</tr>
<tr>
<td>4</td>
<td>Program Risk List</td>
<td>57</td>
</tr>
</tbody>
</table>
An Effective Specification Development Algorithm

The first order of business on any new program is the identification of requirements that collectively define the problem to be solved. This work can be completed in a timely and affordable way producing a quality definition of the problem space in terms of the minimum collection of requirements each one of which defines an essential characteristic of the system, or item thereof, to be developed. The target we should shoot for is all of the essential characteristics identified (no missed essential characteristics) and no unessential characteristics identified. Success in this process can be encouraged by the enterprise developing a set of specification templates and a corresponding set of data item descriptions coordinated with models selected to accomplish the requirements work and an effective suite of tools within which to capture the results. Every requirement that appears in every specification should be traceable to a model artifact from which it was derived. Before embarking on a new program, the enterprise needs to have selected this preferred set of models and tools and trained its staff to employ those models effectively entering the results in the tools suite. Every specification released is printed from the tools suite and should pass through a formal review and approval process as should any subsequent changes to any specification. As the models create work products, commonly simple diagrams, they should be saved in an organized fashion in paper or computer medias and formally released so as to be available for future system phases and modification projects.

Figure 1 offers a view of the overall process within which the preparatory work and definition activity that is discussed in the first section falls. Before entering a program, the enterprise should have prepared itself for the requirements work that will have to be done. The enterprise needs models to apply for the cases where the product is going to be implemented in computer software and in hardware. Both cases are covered in this tutorial followed by a discussion of integrated modeling.

![Figure 1 Overall Process](image-url)
1. Specification Templates and DIDs

Many system development organizations experience some difficulty in clearly identifying appropriate requirements for inclusion in specifications they must develop and they find it difficult to accomplish the work in an affordable and timely fashion. Over a period of the last two years the author developed an algorithm for improving system development organization ability in this work using templates and specially developed data item descriptions (DIDs). It requires some work to prepare the functional organization to support programs and some work on the part of proposal teams to accomplish initial analyses and extend the templates made available from the functional system engineering department to provide program-ready data, and work by the program teams starting with contract award and running through the period of time while specifications are being developed.

The goal of this specification algorithm is to provide for affordable and timely enterprise and program definition and documentation of new product technical requirements, the management and maintenance of related data, and publication and subsequent configuration control of the resulting documents.

The work required to implement the algorithm can be described in three preparatory steps: (1) enterprise engineering work, (2) system engineering generic work, and (3) proposal team work. The first two steps are illustrated in Figure 2. The numbers in the blocks coordinate with the steps in the algorithm. Recommended functional department responsibility for accomplishing the indicated tasks is noted at the lower left corner of the task blocks.

![Figure 2 Preparatory Steps](image)

1.1 Enterprise Engineering Work

Identify and staff an enterprise integration team (EIT) that is responsible for engineering the enterprise common process and acting as the process integration and optimization agent during its development and implementation on programs. The EIT should report to the enterprise executive.
1.1.1  The enterprise, through the efforts of the EIT, must develop a common process diagram that generically identifies all program work at a level of indenture that is adequate for making clear what work must be commonly done and allocating the corresponding work responsibilities to functional departments responsible for supplying programs with the necessary resources to accomplish that work well.

1.1.2  Allocate all work on the common process diagram to functional departments forming a task allocation matrix. This matrix establishes the requirements work that functional departments must be prepared to do on programs and any training that the functional departments are funded for and capable of performing must be focused on these tasks. This matrix covers the whole enterprise capability but in this section we are focused on doing the requirements work.

1.1.3  Functional departments collect all work allocated to them and build department manuals that explain what work must be done and provide links or descriptions for how to perform this work. One of the departments will be system engineering that will have responsibility for specification development and management on programs. For each task a functional department has responsibility, that department must identify work products that will result as a function of having accomplished the work on a program. EIT must integrate and optimize the evolving functional department work descriptions and work product identifications to ensure overall efficiency and effectiveness. All work products must have at least one user. Work products must be linked to the common process diagram tasks. Specifications are an example of a task work product and the work product of interest in this algorithm.

1.2  System Engineering Generic Work

1.2.1  To the extent that work products are documents, the responsible functional department must prepare a template containing the basic structure of the document in terms of generic paragraphing structure and calls for graphical images. In preparing for implementation of specification development and management work on programs in general, the functional system engineering department will select the specification standards that will be applied respecting the common customer base of the enterprise. They will review these standards and associated data item descriptions (DID), ensure that the system engineering department manual adequately covers specification standards the enterprise has chosen to respect, and build a set of specification templates (paragraph numbering and titles only), one for each kind of specification that will commonly have to be prepared on programs.

1.2.2  For each specification template defined, the system engineering department will determine one or more preferred modeling approaches for each kind of requirement in the template. The modeling approaches encouraged are the following relative to the primary kinds of specifications that will have to be developed:

- System Specification
- Hardware Performance Specification
- Software Performance Specification, General
- Software Performance Specification, Database
- Software Performance Specification, DoD IS
- Traditional Structured Analysis
- Traditional Structured Analysis
- Unified Modeling Language (UML)
- IDEF-IX
- DoDAF
1.2.3 Map models to templates such that for any of the specification s listed above there is a model identified for each paragraph of each specification. Ideally, all of the paragraphs of a particular kind of specification would employ models from the same family as suggested in the pairing above.

1.2.4 Map specialty engineering disciplines to specification templates in preparation for program mapping of these disciplines to specific product entities as a means of directing specialty engineering requirements analysis work.

1.2.5 For each template and model combination, prepare a DID communicating how the analyst will prepare the specification using a particular modeling approach and template. The DID must clearly show the connection between the modeling artifacts and the template paragraphing structure. The DID paragraphing structure must coordinate with the modeling components that are intended to yield derived requirements.

1.2.6 Map the functional departments that will be responsible for performing requirements analysis on programs to the template paragraphing structure for each kind of specification telling where the programs will acquire the analysts to accomplish the specification development work.

1.2.7 Prepare a template and DID for a system definition document (SDD) within which program structured analysis work products will be captured and configuration managed. These work products are to be captured in appendices of the document. An alternative is to capture the modeling artifacts within a computer tool that can be configuration managed.

1.2.8 Map the appendices of the SDD to the paragraphing structure of the templates telling in what appendix the corresponding work products will be captured.

1.2.9 Combine the functional department map (1.2.5), models map (1.2.3), and SDD appendix map (1.2.7) on a single matrix for each template and make these matrices available for program use.

1.2.10 EIT and the functional system engineering department must cooperate on selection of one or more computer tools or paper and pencil algorithms with which to accomplish requirements analysis on programs. Built a generic schema coordinated with preferred methods and models.

1.3 Proposal Work That Prepares the Program for Initiation

Proposal team work is illustrated in Figure 3. Blocks that do not have numbers coordinating them with the steps in this algorithm are not covered by the algorithm because they are not directly related to requirements analysis and specification development but these blocks add valuable context.
1.3.1 When beginning the proposal or program work, the manager should establish a program integration team (PIT) staffed by engineering, manufacturing, verification, logistics, and quality and a program business team (PBT) staffed by finance, contracts, scheduling, business information systems, and administration. Both of these teams should report to the program manager. These two teams could be combined as a staff function to the proposal manager but they will have integrating and optimization roles to play across the product oriented teams.

1.3.2 The PIT will perform an initial system analysis that will result in a clear understanding of any requirements provided by the customer, formatting of those requirements into alignment with the enterprise DID for a system specification, and adding to those requirements the results of their own system analysis work. A system environmental requirements analysis activity will expose a set of tailored standards covering the natural environment corresponding with spaces within which the system shall have to operate. A threat analysis will lead to exposure of hostile requirements. An interface analysis will identify external and top level internal interfaces that will be characterized in requirements statements. The result will be a preliminary system specification for submission with a proposal. If possible, this analysis work will be continued to develop all of the immediately subordinate specifications each of which will be the development responsibility of one of the top level integrated product and process teams (IPPT) to be identified and staffed subsequent to contract award.

1.3.3 The modeling work described in paragraph 1.3.2 will yield modeling artifacts from which requirements may be derived. The preliminary system specification development and any other specifications developed during the proposal effort will follow the pattern described in the
next paragraph. The second tier specification development may be delayed until a contract award but should precede the formation of the top tier IPPT. In all cases, requirements are derived from a model.

1.3.4 Requirements flowing from the structured analysis work will flow into a requirements analysis sheet (RAS) implemented in a computer database tool. All requirements entered into the RAS must include a traceability reference to the model from which they were derived.

1.3.5 Out of the initial PIT system analysis work will also come the preferred product entity breakdown diagram upon which the PIT, working in concert with integrated business team personnel, will construct overlays for organization responsibility breakdown (IPPT assignments), specification tree breakdown, engineering drawing breakdown, work breakdown (WBS), and manufacturing breakdown. The work breakdown will be handed off to the business team that will use it as the basis for building the program work definition, cost estimate, and IPPT work budgets. The IPPT will be assigned so as to align perfectly with the WBS making it possible to present to each IPPT leader as the teams are formed, a copy of the top level specification for which the team is responsible and the related budget and planning package for the whole WBS the team is responsible for as well as their top level schedule responsibilities encouraging the result that the IPPT leader may be held accountable for managing all aspects of the development of the entity assigned to the team.

1.3.6 The PIT will select a set of templates that correspond with the kinds of specifications that will have to be prepared on the program and the related DIDs that are coordinated with the models that will be applied in the analytical work. The PIT must also map specialty engineering disciplines to product entities to aid teams being formed to staff appropriately.

1.3.7 The PIT must take action to cause adequate computer tool seats to be allocated to the proposal team and accomplish any planning necessary for the subsequent program relative to the use of any requirements database tools and make any needed adjustments in database schema for the program.

1.3.8 The PIT shall capture the results of structured analysis work performed during the proposal activity in a preliminary system definition document (SDD) that will be used as the basis for subsequent lower tier analyses.

1.3.9 Any specifications developed in the proposal effort must be formally reviewed and approved by the proposal manager.

1.4 Work Subsequent to Contract Award

Specification related work to be accomplished subsequent to contract award is illustrated in Figure 4. This work is repetitious in nature progressively working down through the expanding architecture. Top-level teams may shred out during program work yielding sub-teams but in all cases, the top-level teams are responsible as managers for all lower tier team activity. This telescoping management responsibility is applicable throughout the team structures. During program performance, lower tier requirements analysis responsibility may be passed down
through the team structure with immediately superior team reviewing and approving of all immediately lower tier team specifications or the responsibility may be retained by the higher-level team but these decisions must be coordinated with the team budgets and staffing considerations arrived at during proposal work.

Figure 4 Program Work

1.5 The Preferred Templates

Ideally, the development organization would build a set of templates (paragraphing structure with no content) and data item descriptions (DID) that tell how to build a specification following the related template using a particular set of models. These should be maintained by the functional department in system engineering that has overall requirements and specification work responsibility. These should be available for reference or download by any new program.

Figure 5 offers a view of the preferred template for a system or hardware specification using traditional structured analysis as the modeling choice. The template is annotated with the preferred modeling artifact that will be used to identify the corresponding requirements and the functional department from which the program will obtain personnel to accomplish the related modeling work using the Figure 3 organizational structure. The data item description (DID) acronym in the model column means that the content is driven by the content of the DID used as the basis for the specification. The department references are cut at a very high level in this case and should be identified at one or two layers below this level but Figure 3 goes no deeper. The APP column gives the Appendix in the System Definition Document where the work products can be found.

The structure in Figure 5 can also be used for computer software requirements specifications (SRS) with paragraph 3.1 rewritten for UML and the model column updated to reflect UML artifacts.
<table>
<thead>
<tr>
<th>PARA</th>
<th>TITLE</th>
<th>MODEL</th>
<th>DEPT</th>
<th>APP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>SCOPE</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>2.</td>
<td>APPLICABLE DOCUMENTS</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.</td>
<td>REQUIREMENTS</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1</td>
<td>Functional and performance requirements</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.1</td>
<td>Missions</td>
<td>Mission Analysis</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.2</td>
<td>Threat</td>
<td>Threat Analysis</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.3</td>
<td>Required states and modes</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.3.1</td>
<td>Functional analysis</td>
<td>DID</td>
<td>2100</td>
<td>A</td>
</tr>
<tr>
<td>3.1.3.2</td>
<td>Subordinate entities</td>
<td>DID</td>
<td>2100</td>
<td>C</td>
</tr>
<tr>
<td>3.1.3.3</td>
<td>Interface relationships</td>
<td>DID</td>
<td>2100</td>
<td>D</td>
</tr>
<tr>
<td>3.1.3.4</td>
<td>Specialty engineering requirements</td>
<td>DID</td>
<td>2100</td>
<td>E</td>
</tr>
<tr>
<td>3.1.3.5</td>
<td>Environmental model</td>
<td>DID</td>
<td>2100</td>
<td>B</td>
</tr>
<tr>
<td>3.2</td>
<td>Entity capability requirements</td>
<td>Functional Analysis</td>
<td>2100</td>
<td>A</td>
</tr>
<tr>
<td>3.3</td>
<td>Interface requirements</td>
<td>N-Square Diagram</td>
<td>2100</td>
<td>D</td>
</tr>
<tr>
<td>3.3.1</td>
<td>External interface requirements</td>
<td>N-Square Diagram</td>
<td>2100</td>
<td>D</td>
</tr>
<tr>
<td>3.3.1.m</td>
<td>External interface m</td>
<td>N-Square Diagram</td>
<td>2100</td>
<td>D</td>
</tr>
<tr>
<td>3.3.2</td>
<td>Internal interface requirements</td>
<td>N-Square Diagram</td>
<td>2100</td>
<td>D</td>
</tr>
<tr>
<td>3.3.2.m</td>
<td>Internal interface m</td>
<td>N-Square Diagram</td>
<td>2100</td>
<td>D</td>
</tr>
<tr>
<td>3.4</td>
<td>Specialty engineering requirements</td>
<td>DID</td>
<td>2100</td>
<td>E</td>
</tr>
<tr>
<td>3.5</td>
<td>Environmental conditions</td>
<td>3-Layered Env Model</td>
<td>2100</td>
<td>B</td>
</tr>
<tr>
<td>3.6</td>
<td>Precedence and criticality of requirements</td>
<td></td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>4.</td>
<td>VERIFICATION</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>PACKAGING</td>
<td>DID</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>NOTES</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>6.1</td>
<td>Requirements traceability</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>6.1.1</td>
<td>Inter-specification specification traceability</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>6.1.2</td>
<td>Verification traceability</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>6.1.3</td>
<td>Modeling traceability</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>6.1.4</td>
<td>Section 2 traceability</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>6.1.5</td>
<td>Programmatic traceability</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>6.2</td>
<td>Glossary</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>6.3</td>
<td>Specification maturity tracking</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>A.</td>
<td>APPENDIX A</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
</tbody>
</table>

Figure 5  System and Hardware Specification Template

A similar mapping should be provided for the software requirements specification (SRS) based on, in the author's view, the use of UML. Customers often require conformance to a DID supplied by them but may permit tailoring. The outline included in Figure 5 is a significantly tailored version of MIL-STD-961E to group all requirements so as to correspond with the modeling components contained in traditional structured analysis described in paragraph 2.2.1. k of the standard. The author believes this format will work with software as well but a different DID is required coordinated with the modeling approach selected (UML encouraged).
1.6  Modeling Work Product Capture Document

Programs should also be provided with a template for a System Definition Document (by whatever name) within which they can easily capture the results of all modeling work so that it can be preserved beyond the period of time when that work is actively being pursued. A later section describes this document. The appendices of the SDD are referred to in Figure 5 in the APP column and explained in Figure 20 and related text.

2  Structured Analysis

There are many models that can be used to accomplish an organized requirements identification effort that is preferred to an ad hoc method because it will most often hit the target noted earlier - identification of all of the essential characteristics and identification of no unessential characteristics. This process description encourages the use of traditional structured analysis in the near term to develop and identify the requirements for systems, hardware, facilities, real property improvements, and personnel actions captured in procedures as discussed in paragraph 2.1. One of the most difficult tasks in system development revolves around the relations between the system entities, the interfaces. This discussion of traditional structured analysis also contains a complete algorithm for identifying all system relations. Where the product is going to be implemented in computer software, unified modeling language (UML) is encouraged within a process context offered in paragraph 2.2. It is intended that a development organization should develop a process transformation roadmap needed to transition from this mixed method for the modeling work to a universal modeling approach, still evolving, that can be applied to all requirements work at the earliest possible. See paragraph 2.3 for a discussion of integrated modeling.

2.1  Traditional Structured Analysis

Figure 6 illustrates an overview of the traditional structured analysis process (TSA). The eleven numbered steps are briefly introduced followed by additional details in subordinate paragraphs.

1. Understand the User Requirements - Through conversation with the user and/or reading a user requirements statement or specification, the developer tries to understand the user's need. This is not ever easy because the user is able to explain only what their mission interests are and the developer needs hard engineering data.
2. Decompose - Users commonly have problems that are too grand to be easily understood in a single small document or simple diagram. These problems commonly have to be decomposed or partitioned into a collection of smaller related problems.
3. Functional Flow Diagram - The TSA approach employs some form of functional analysis as the decomposition medium.
4. Performance Requirements Analysis - The functions are translated into performance requirements.
5. Requirements Analysis Sheet (RAS) - The strings of functions, performance requirements, and product entities to which they are allocated appear as rows in the RAS. The RAS also is used to capture all system requirements linked to the model from which they were derived.
6. Requirements Allocation - Performance requirements are allocated to product entities in the RAS.
7. Product Entity Structure - The physical and logical entities that comprise the system are arranged in a hierarchical structure that is used as the basis for WBS, specification identification, team assignments, and many other program applications.
8. N-Square Diagram - The interfaces that must exist between the product entities are identified through a pair-wise analysis of all possible interface relationships.
9. Environmental requirements for the expanding product entities are determined through application of a three-layer model.
10. Specialty engineering requirements are identified by a group of specialist linked to the product entity structure by a specialty engineering matrix.
11. This process is applied iteratively as lower tier entities are identified through functional analysis.

Figure 6  Overview of the Traditional Structured Analysis Process
2.1.1 A System Defined

A man-made system is a collection of entities that are meant to interact in predictable ways with an environment and with each other via relations between them to achieve a useful function identified and articulated by a customer as a mission need statement. Therefore, systems are composed of entities and relations between the entities. The system is intended to satisfy the mission need statement, the system’s ultimate function, depicted on system diagrams as a rectangular block titled System Need and identified with a functional identifier F. The need is allocated to the system depicted on system entity model by a rectangular block named “system” (or a particular system) and identified with a product entity identifier A.

A system interacts within an environment as shown in Figure 7. The environment for every system is everything in the Universe (U) less the entities that are part of the system product entity structure (Q = U - A). One can reduce the scope of the environment to those elements that will have some influence on the system. The line joining the system and environment in Figure 7 (I2) indicates the relations between the two (external interfaces). The line joining the system on both terminals (I1) indicates the internal relations between system entities (internal interfaces) yet to be defined within the system.

![Figure 7 Ultimate System Diagram](image)

2.1.2 The System Environment

The environment for any system is composed of the subsets illustrated in Figure 8. While all environmental effects on the system are relations, they may be partitioned between those that are commonly considered environmental stresses and the cooperative environmental elements that are treated as external interfaces commonly developed by a pair of teams or contractors responsible for the terminal product entities.

A context diagram, such as that shown in Figure 9, even though similar in nature to Figure 7, offers a useful simple model for focusing attention on identifying all external relations. Some of these terminators will be natural, non-cooperative, or induced environmental stresses. Others include hostile stresses determined through a system threat analysis as well as both stresses and useful relations with cooperative systems. Though this diagram was conceived as the beginning of the modern structured analysis model, it has a useful purpose in TSA as a means of viewing all of the inside-outside relationships and in UML as an organizer of use cases. It can be used to make a first impression in the line I2 in Figure 7.
The system natural environment is determined by defining all of the spaces within which the system will be employed based on an analysis of the intended mission and basing concept. The spaces are coordinated with a set of environmental standards. Each standard is studied for necessary content and the remainder tailored out. Each selected parameter is then studied for an appropriate range. The system natural environment is then the union of the selected parameters from the selected standards.

The non-cooperative environment is defined by determining what stresses will be applied to the system from man made systems which are neither hostile nor cooperative. An example of non-cooperative stress is electromagnetic energy. Self-induced environmental stresses are not easily
determined at the system level because one needs to understand energy sources and other stressors within the system determined as part of the design of end items.

System cooperative environmental relations are defined by determining how the system to be developed will associate with other friendly systems already in existence or being developed. These associations may be coupled into or out of the system in terms of information, physical association, materials, or energy.

2.1.3 System Functionality

A function is a necessary activity for a system to perform. It may be static, dynamic, or both. It should be named using an action verb followed by a noun. A function is depicted in modeling the system as a rectangular block identified by an action verb name centered in the box and a function identifier (ID) in the lower right corner. The ultimate function for any system is the customer need the name of which is the need statement possibly paraphrased to fit into the space provided coupled with a function identifier F.

Two or more functions can be linked together using directed line segments to show a sequence of functions. In Figure 10 the understanding is that function F1 must be accomplished before function F2. Combinatorial symbols may be added to permit more complex sequential relationships. The combinatorial symbols encouraged are AND, inclusive OR (IOR), and exclusive OR (XOR) with the common logical meanings. Enhanced functional flow block diagramming adds loop (repeat a function until a specific outcome has been attained) and iterate (repeat a function a specific number of times) combinatorial symbols that can be useful. Diagrams so constructed are called functional flow diagrams. These diagrams may be oriented on the page with their primary flow axis arranged horizontally or vertically with the flow in either direction.

![Figure 10 Function Sequence](image)

For any function with identifier F@ (where @ is a string of length n (including n=0) composed of characters from the set {A through Z less O)U(a through z less l)U(0 through 9)}) there may exist one or more subordinate functions F@# (where # is a single character from the same set identified above which differentiates other functions at that level from one another). This is illustrated in Figure 11. Every function need not have an expansion. There is no need to assign function identifiers in alpha numeric sequence on a page of the diagram but it helps the human to use the diagram if they are assigned initially coordinated as much as possible with the function sequence. If a function is deleted subsequent to a release of the diagram, that identifier should not be used again. If the number of functions on any one diagram exceeds the maximum number of symbols available, 60, change the diagram to reduce the number to less than 60.
Ideally, whoever accomplishes the initial analysis of the need, would do so using the functional analysis process described here where the first decomposition is the system life cycle as shown in Figure 12 and the second is an expansion of the life cycle function “Use System” (F47) to expose the top level operational intent and initial content of the user requirements documentation or preliminary system specification. If the customer or other initial analysis agent applies an unstructured or ad hoc approach, then the development organization may have to accomplish a functional analysis and try to map the requirements identified by the customer (user and acquisition agent) to the functionality exposed when they do accomplish this work.
Ideally, the development organization would extend the functional analysis into the remainder of the system life cycle functions as well as the Use System function determining appropriate resources for the process steps of the development program and the product system being developed such that the physical product delivered will be jointly optimized relative to its product and process. Commonly, process functions do, or should, influence product functions and the corresponding product entities needed as well as the opposite case.

2.1.4 Performance Requirements Derivation and Allocation

The functions identified in the functional flow diagram must be translated into performance requirements that tell what the system and its parts must do and how well it must do them as shown in Figure 13. These statements can be first developed as primitive statements for example phrased as “Velocity ≥ 600 knots” without complete sentence structures and subsequently transformed into complete sentences in the chosen language. Traditionally, a requirements analysis sheet (RAS) has been used to capture the function identification, the primitive performance requirements statements, and the allocation of these performance requirements to product entities. One could allocate the function names directly to architecture but often one finds a one-to-many allocation result this way whereas allocation of performance requirements tends to follow a one-to-one pattern. To fully characterize a function it may require identification of multiple performance requirements and these several performance requirements may be allocated to different product entities.

The RAS as traditionally used is incomplete, unfortunately, and this discussion will use a RAS complete. We will show how the three kinds of constraints can also be captured in the context of Figure 13 shortly. The intent is to be able to capture all requirements in a RAS linked to a modeling artifact implemented in a computer application.

<table>
<thead>
<tr>
<th>FUNCTION</th>
<th>PRODUCT ENTITY</th>
<th>REQUIREMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>MID</td>
<td>NAME</td>
<td>PID</td>
</tr>
<tr>
<td>F123</td>
<td>Provide Thrust</td>
<td>A12</td>
</tr>
</tbody>
</table>

Figure 13 Traditional Requirements Analysis Sheet

2.1.5 Product Entity Structure

System functionality is accomplished by physical entities that form part of the physical system. These entities in the system, in the aggregate, comprise the product entity structure that the author formerly referred to as the system architecture. The word architecture has taken on a significantly broader meaning in recent years convincing the author that what he formally referred to as architecture should now be referred to by the more isolating term product entity.
structure. The function F is accomplished by product entity structure element A, the system, by definition as shown in Figure 7. Lower tier entities should be exposed following Sullivan’s encouragement of form follows function. Trade studies may be appropriate to make hard decisions on the best implementation of a particular function in early program phases. The physical entities that accomplish functionality can be partitioned into five classes: (1) hardware, (2) computer software, (3) facilities and improvements to real property, (4) procedural definition, and (5) humans following procedures or acting autonomously. We could merge the last two into one. The relationship between functions, the physical entities in the product entity structure, and the corresponding performance requirements is depicted on a simple requirements analysis sheet as shown in Figure 13.

The aggregate product entity structure for a system is illustrated in a hierarchical model connecting the product entities arranged into a breakdown block diagram as illustrated in Figure 14. The entity identifiers follow the same pattern as the function identifiers explained earlier beginning with a letter A because of the author's previous use of the word architecture for this view. The entities stream out of the RAS available for structuring into the product entity block diagram. Ideally, this work would be accomplished by a team of people representing hardware and software engineering, manufacturing, procurement, and verification with strong system/program leadership by the PIT. Initial functional analysis and allocation must concentrate on understanding user mission needs. This will generally require intense interaction with the user, ideally using system models to encourage mutual understanding. Alternative ways of implementing functionality with different product entities should be considered and where the decision is very difficult they should be subjected to a trade study.

Figure 14 Product Entity Structure
2.1.6 Allocation Pacing Alternatives

The conduct of the functional analysis and allocation work can be paced in one of four different ways:

(1) Instant – As soon as functions and/or their corresponding performance requirements are identified, they must be immediately allocated to the expanding product entity structure.

(2) Terminal - All of the functional analysis must be complete before any functions and/or their corresponding performance requirements can be allocated to product entities.

(3) Layered - The analyst completes one layer of the expansion of a function and must allocate all of the exposed functionality and/or the corresponding performance requirements to product entities before further expanding that function. This works best if all product entities related to a layer are defined in terms of their requirements and design concept before pursuing the next functional layer and its allocation but there generally is not sufficient time available and one must accomplish a lot of this work in parallel leading to some risk and interaction. The layered approach has been popularized by Mr. Bernard Morais and the late Dr. Brian Mar under the title FRAT.

(4) Progressive – The analyst completes several layers of the functional analysis without allocating any of it to product entities. At a layer guided by experience the analyst begins allocating performance requirements derived from higher tier functionality to product entities. Design concepts are defined for the product entities at the higher levels and these act to both guide and constrain lower tier functionality progressively. Allocation is delayed throughout the analysis such that higher tier design concepts help to steer lower tier functional analysis tending to isolate iteration to one structure (functionality, architecture, or allocation) at a time. The two extreme cases (1 and 2) are flawed due to a need to iterate F, A, and allocations excessively in the case of 1 and a common need to significantly change lower tier functionality after the higher tier allocations begin in the case of 2. The progressive approaches either by layer or guided by experience will generally produce better results with less modeling hysteresis.

There exists a downward limit in decomposition or expansion of the functional portrayal. This limit for any one branch in the expansion is best determined on the product entity plane based on the analyst’s understanding of the problems related to the lowest tier product entities. Where all of the lowest tier entities are fully characterized supporting procurement or in-house design, the functional analysis work can be reduced to maintenance of the models and related data. There is one more layer of requirements related to parts, materials, and processes but that is driven by design decisions during synthesis and commonly has already been completed by those responsible for the sources of these entities. The logistics analysis process may require a switch to a process diagram and if the progressive allocation approach described has been followed the functional flow will have been migrated toward a process diagram at the lower tiers.
Figure 15 illustrates a geometric view of the process of deriving performance requirements from functions and the allocation of those performance requirements to product entities. For example, a performance requirement has been derived from F4713 and allocated to product entity A11. This plane represents the normal RAS used only for function allocations. As suggested by the other structures we will use the geometric structure to expand the RAS to cover all requirements.

2.1.7 System Relations

As the product entity structure is formed, the analyst can begin to identify the needed internal relations between the system entities by using an n-square diagram where the product entities are identified down the diagonal at some level of indenture. For a given analysis where the number of entities being studied for interface relations in an n-square diagram is N the largest possible number of relations is N(N-1) counting each direction as one possibility between each pair of entities. Interfaces between these entities is pre-determined by the way that functionality is allocated to the entities. Therefore, one may explore the list of function-product entity pairs.
associated with each product entity in the n-square diagram. This is referred to as a pair-wise analysis of the n-square diagram intersections.

Figure 15 includes an n-square diagram with only half of the square showing (the remainder hiding behind the other structures in the diagram). The diagonal (containing the product entities we are interested in accomplishing the pair-wise analysis on) has been aligned with the product entity axis of the function-product entity matrix (which corresponds to the simple RAS of Figure 13).

The process for marking the intersections of the function driven relations matrix (n-square plane) entails a pair-wise analysis of the function-product entity pairing relationships marked on this matrix. Interface Ix is encouraged by the conclusion that if F4711 maps to A13 and F4712 maps to A11 then there is a possible demand for an interface between A11 and A13 to implement that relationship. So we map functions to product entities but we map F-A pairs to interfaces and those interfaces are pre-determined by the way we allocate functions to the product entities.

If the system is a modern war ship or the system that will take man to Mars, a pair-wise analysis of the function driven relations matrix would be beyond human comprehension if attempted all at one time. We can, however, partition the analysis work to one interface expansion at a time and it is not so overwhelming. At any one level of product entity granularity, we can explore one layer of product entity structure expansion for internal interfaces within the parent item. If there are five subordinate entities then the number of possible interfaces to be examined in a pair-wise fashion would be 5x4 or 20. Similarly, external interfaces can be analyzed individually. Of course, it will still be necessary to accomplish considerable interface integration work because of the partitioned process. This process will go very fast if the analyst is very familiar with the problem space and the evolving solution concepts. The complete algorithm is extended in subsequent paragraphs.

The requirements analysis sheet (RAS) identifies every possible pairing of functions and architecture entities. We may sort this listing so that all of the functions (or performance requirements derived from those functions) allocated to each entity are grouped by entity. Then we can pile up the allocations onto the product entity squares on the diagonal of a physical n-square diagram as suggested in Figure 15.

The discussion so far has dealt only with internal interface identification all defined on the function driven relations matrix of Figure 15. To cover external interfaces we add the larger n-square diagram on Figure 15. The diagonal in this case includes all of the product entities plus all of the external entities in the cooperative environment. We can identify relations between these external and internal entities in the same way we did the internal ones. The association of function F4712 to cooperative environment entity QC1 and allocation of a function to A12 may define a need for an interface I2y.
2.1.8 Environmental Relations Algorithm

2.1.8.1 System Environmental Relations

The system environment consists of all entities in the Universe less those that are in the system. That is, \( Q=(U-A) \) where \( Q \) is the environment, \( U \) is the Universe, and \( A \) is the product entity structure of the system being developed. It is convenient to partition all system level environmental relations into the sets illustrated in Figure 8. The system cooperative environment (QC) can actually be treated as an external interface and can be developed using the algorithm covered in Paragraph 2.1.7 very effectively. External cooperative systems are simply located on an extension of the product entity axis forming the cooperative environment axis. The hostile environment (QH) can best be understood through analysis of threats posed by hostile forces. The non-cooperative environment will yield to the same thought process applied in the threat analysis except that the stresses applied to the system are not applied for hostile purposes, rather simply because the system being developed is sharing a common operating space with other systems. Electromagnetic interference is an example of the stresses applied in this set.

System time (QN2) is studied using time lines oriented about the functions that the system must satisfy. When we allocate those function (or their corresponding performance requirements) to architectural entities the timing requirements corresponding to the functions are applied to the entity as timing requirements.

System space (QN1) is defined through mission analysis such that it is determined in what volumetric spaces of the Earth (surface, subsurface, and aerodynamic) and/or surrounding space and/or distant bodies the system shall function within, on, or around. For each space, that space is teamed up with one or more natural environmental standards that define that space. Each of these standards is then studied to determine which natural environmental (QN3) parameters included in the standard shall apply to the system being developed. Those that do not apply are tailored out of the standard. The selected parameters are then studied to ensure that the range of values is appropriate for the system. If the range in the standard is too broad it is tailored to narrow the range of values to that for which the system shall be designed. This process is repeated for each standard linked to a system operating space.

Figure 15 also extends the RAS concept to include environmental requirements analysis. The system environment as depicted in Figure 7 is illustrated at the diagram “origin” on the environmental axis that happens to coincide with the architecture “origin” that corresponds to the whole physical system A.

2.1.8.2 End Item Service Use Profile

An end item is a major element of a system that generally retains its physical configuration throughout its mission performance and has an end use function. The end items may remain fixed in place during use or move over great distances and maneuver within the system spaces as a function of the system mission and the end item’s application in the system. Each end item should be designed to endure only those natural environmental stresses anticipated so it is necessary to determine what subset of the system natural environment each end item shall be
exposed to. To accomplish this, one must create a physical process flow diagram. This is not the same as the functional flow diagram used to identify system architecture and performance requirements. The blocks on a functional flow diagram represent things that have to happen whereas the block of a process diagram represent a real world analogy. You cannot profitably consider system entities flowing through the functional flow diagram, indeed we are using the diagram to determine what those entities should be. However, we can imagine the system product entities flowing in the process diagram where each process acts as a transformation on the system entities.

The first step in defining end item environmental relations is to map the system environmental parameters to the process steps at some level of indenture, generally at a level where the environmental map does not change significantly below that process level. This work defines an environment for each process. The next step is to map the architectural entities to the process blocks. If an entity only maps to a single process step, it simply inherits the process environmental set. If, as is so often the case, the architectural entity maps to two or more process blocks, then it will be necessary to apply some kind of integrating process to any differences in environmental stresses and their values observed between the two or more process blocks. The rule most often selected initially is to pick the worst-case range for each parameter across the process values being evaluated. If this rule does not adversely influence system cost, then it is an adequate solution. If this approach either results in an adversely narrowed system solution space, then an alternative to “worst case” must be derived. Often time lines will show that the problem environmental stress will be applied over such a short time as to be insignificant. In other cases, one may find that the problem can be narrowed to some particular combinations of values that are very unlikely to occur. If the problem is intractable, it may be necessary to restrict one or more system environmental parameters more severely than is currently being done. Generally, this will have an adverse effect on system performance.

The self induced environment (QI) can best be studied and defined at the end item level since it is end items which contain the sources of energy and other stresses of interest which will reflect commonly through a natural environmental parameter right back into the system. Since the self-induced stresses are commonly greater in magnitude than the corresponding natural stresses for that same parameter, these induced relation values have the effect of extending the range defined through the application of the end item service use profile algorithm discussed above.

2.1.8.3 Component Environmental Relations

The environmental relations appropriate for components installed within end items are simply the end item environmental stresses if the end item has no altering effect on those stresses and all spaces within the end item offer the same environmental stresses to components installed within them. Where an end item does have a modifying effect on end item stresses but all spaces within and upon the end item offer the same stresses, it is necessary to determine the end item design effect on end item environmental stresses and the result is the set of installed component environmental stresses. The most complex case occurs when the end item must be partitioned into two or more spaces more often called zones of common environmental stresses. The value of each end item parameter must be determined for each zone thus defining the environment for each zone. Then one maps the components to the zones and they inherit the zone environments.
Commonly, the job is not complete at this point because it is found that there is no zone within which one or more components can be installed in a particular end item that will cause their environmental stress limits to be satisfied. When this happens, it is necessary to either change the component environmental specification values or include an environmental control system as an added entity into which the components with the environmental range shortfall problem are placed.

2.1.8.4 Environmental Impact

The environmental effects discussed in the three previous cases deal with environmental stresses the environment will apply to the system but there may be cases of the opposite direction that will cause damage to the environment. Once identified by someone skilled in environmental impact, these can be treated like safety hazards to be mitigated through re-design, procedural controls, or compensating environmental actions. In the case of military systems it is very difficult to mitigate the damaging effects of warfare but these systems can also have damaging effects on the environment in peaceful use such as training and maintenance. Often these materials are just naturally dangerous to be around as illustrated in the difficulties observed in the base closure efforts where many adverse environmental effects have had to be identified and mitigated.

2.1.9 Specialty Engineering and RAS Complete

The system engineering agent for the system must build a list of all of the specialty engineering disciplines that will be applied in the development of the system. A specialty engineering scoping matrix should be prepared between specialty engineering disciplines that may be required in development of the system and the product entities. This will help to determine team staffing needs in that area and connect people in those disciplines with a need to do specialty engineering requirements analysis for the indicated items. Figure 16 adds one more plane, the specialty engineering scoping matrix, to the construct previously illustrated in Figure 15 providing for allocation of specialty engineering disciplines listed to architecture.

Specialty discipline H7 is shown mapped to architecture item A11. This must be followed by analyst definition of one or more discipline H7 requirements that will flow into the specification corresponding to the product entity. The structure exposed in Figure 16 is a complete RAS showing all of the important requirements related relationships supporting the requirements analysis process leading to the identification of every kind of requirement appropriate to the system and hardware specifications and all of them linked to and derived from a model.
The fact that an aircraft airframe will have to be checked for alignment during manufacture and after hard use (hard landing or pulling excessive g’s in flight, for example) identifies a need for a relation between the airframe and the equipment which will be used to accomplish the alignment. Today this will generally call for some form of laser optical application so the airframe would have to either include targets, detectors, or mirrors or provisions for these to be applied. The manufacturing and maintenance engineers would have to consider all of the ways there may have to be relations between support equipment and the operational entity. There may be other cases where these disciplines have to call for internal relations within the system entities. For example, if the system must include built in test (BIT), there will have to be relations between most of the on board equipment and some entity that concentrates the BIT effects for display to operations and/or maintenance personnel.
The needs of operations personnel, such as aircraft pilots, locomotive engineers, ship captains, and automobile drivers provide a tremendously rich class of entity relation possibilities. The physical relations are always fairly simple in that the human operator has only his/her senses, mental acuity, and physical strength through which to interact with the system. This problem is made much more complex, however, because not all humans will react in precisely the same way to a particular stimulus. It will be necessary to determine the complete data set that the human will require under all operating conditions and in what way the human shall influence system behavior in terms of controls. Operator sequence diagrams, built like a UML activity diagram with swim lanes, can be useful in doing this work.

Every one of the specialty engineering disciplines selected for the program must be evaluated for entity relation driving potential and those persons doing that work alerted to their responsibilities in identifying relations for further consideration by the whole team.

2.1.10 RAS-Complete in Table Form

The results from the analyses noted in prior text must be captured on its way into program specifications. Certainly, the most advantageous way to do that is in a computer database systems such as DOORS, CORE, or SLATE. Therefore we would expect that some form of tabular structure would suffice. Figure 17 is offered as the candidate view of this table for use during development in capturing the relationship between model, requirements, product, and document entities. The model ID (MID identifies the model from which the requirement was derived. The requirement columns identify the requirement ID (RID) assigned by the computer system for use in traceability. The RID is a made-up computer-assigned unique field using a base 60 numbering system in this example but a hexa-decimal implementation is probably more common. Ideally the requirement statement should be captured in primitive form (attribute, relation, value, and units) wherever possible with different fields for each component of the string. The primitive form is shown concatenated in Figure 17. The final column pair offers specification paragraph number and title.

The sample data included is ordered by model ID alphanumerically separating the data into the four kinds of requirements found in a system or hardware specification. The lists the MID'S respected by the author is still in a state of change as different RAS-Complete possibilities are explored. This may explain the apparent unthinking selection of H and Q for specialty engineering and environmental requirements, respectively. The intent is to identify all possible modeling artifacts with a letter as a source from which every conceivable requirement may be derived. Model letters for UML artifacts have been included in the set and will be introduced later.

This view provides clear traceability between the models from which the requirements were derived and the product entities to which they were allocated for all of the requirements, not just the performance requirements. What the author calls lateral traceability is captured in the database implementing the RAS-Complete. It is also a simple matter to link the rows in the matrix in a database to the corresponding verification requirements as well as the tasks to which they are allocated and their corresponding plans, procedures, and reports. Vertical traceability is, of course, simply a matter of relating the unique RIDs from pairs of requirements in specification parent-child relationships identified by their PID.
MODEL ENTITY | REQUIREMENT ENTITY | PRODUCT ENTITY | DOCUMENT ENTITY
---|---|---|---
F47 Use System | A Product System | A Product System | 3.1.5 Reliability
F47T Deployment Ship Operations | A Product System | A Product System | 3.1.5 Reliability
F47T1 Store Array Operationally | XR67 Storage Volume < 10 ISO Vans | A1 Sensor Subsystem | 3.1.5 Reliability
H Specialty Engineering Disciplines | A Product System | A Product System | 3.1.5 Reliability
H11 Reliability | EW34 Failure Rate < 10 x 10^-6 | A1 Sensor Subsystem | 3.1.5 Reliability
H11 Reliability | RG31 Failure Rate < 3 x 10^-6 | A11 Cable | 3.1.5 Reliability
H11 Reliability | FYH4 Failure Rate < 5 x 10^-6 | A12 Sensor Element | 3.1.5 Reliability
H11 Reliability | GBR4 Failure Rate < 2 x 10^-6 | A13 Pressure Vessel | 3.1.5 Reliability
H12 Maintainability | 6GHU Mean Time to Repair < 0.2 Hours | A1 Sensor Subsystem | 3.1.6 Maintainability
H12 Maintainability | 6GHU Mean Time to Repair < 0.4 Hours | A11 Cable | 3.1.6 Maintainability
H12 Maintainability | U9R4 Mean Time to Repair < 0.2 Hours | A12 Sensor Element | 3.1.6 Maintainability
H12 Maintainability | J897 Mean Time to Repair < 0.1 Hours | A13 Pressure Vessel | 3.1.6 Maintainability
I1 System Interface | A Product System | A Product System | 3.1.5 Reliability
I1 Internal Interface | A Product System | A Product System | 3.1.5 Reliability
I181 Aggregate Signal Feed Source Impedance | E37H Aggregate Signal Feed Source Impedance= 52 ohms + 2 ohms | A1 Sensor Subsystem | 3.1.5 Reliability
I181 Aggregate Signal Feed Load Impedance | E371 Aggregate Signal Feed Load Impedance= 52 ohms + 2 ohms | A4 Analysis and Reporting Subsystem | 3.1.5 Reliability
I2 System External Interface | A Product System | A Product System | 3.1.5 Reliability
Q System Environment | A Product System | A Product System | 3.1.5 Reliability
QH Hostile Environment | A Product System | A Product System | 3.1.5 Reliability
QI Self-Induced Environmental Stresses | A Product System | A Product System | 3.1.5 Reliability
QN Natural Environment | A Product System | A Product System | 3.1.5 Reliability
QN1 Temperature | 6D74 -40 degrees F< Temperature < +140 degrees F | A Product System | 3.1.5 Reliability
QX Non-Cooperative Environmental Stresses | A Product System | A Product System | 3.1.5 Reliability

Figure 17  RAS-Complete in Tabular Form

2.1.11  Traditional Structured Analysis Summary

In summary, a system is defined by identifying its functionality starting with the need (F), allocating that functionality to entities that become part of the system architecture (A). These entities that form the system architecture are selected by determining the performance requirements that the system must satisfy to meet the top-level customer’s need. The pairs of functions and product entity allocations pre-determine how the entities will have to relate to each other through interfaces (I) between the product entities. The environmental elements (E) are defined at the system level in terms of the spaces within which the system must function and the corresponding characteristics of those spaces drawn from appropriate environmental standards covering those spaces. As depicted in Figure 18, the traditional structured analysis effort attempts to define the most cost effective solution such that in N cycles of the process axis of the physical system (generally cyclical in the interest of reuse of system elements) the relation P (process) maps the cross product of the power sets of architecture (A*), interface (I*), and environment (Q*) to the function set F such that F is covered.
For every process $P_i$, there exists a combination of architecture, interface relations, and environmental stresses such that some subset of the function set is covered or accomplished. The power sets of these entities include all of the possible subsets of these entities within their own set thus the power set of $A$ includes every useful combination of product entities relative to every process step. Useless subsets are also included in the power set as well, of course. It is important that the functions be covered in the correct order determined by the sequence of the processes linked together in the process axis. If all of the functions are satisfied in $N$ revolutions of the process axis as planned, then we may say that the system is consistent relative to the use of its product entities, interfaces, and environmental stresses. If there are product entities that are not used in the process or some that are needed but not available we may not have the optimum product entity structure.

This whole process happens in practice somewhat backwards in that for an unprecedented system, one begins the development process only knowing the ultimate function, the need, and must expand everything from that one perspective.

2.1.12 SDD Content and Format

When used to support the application of traditional structured analysis on a program, the preferred SDD format consists of a main body and seven appendices, each providing a capture point for the work products of one of the several fundamental analytical system requirements analysis process areas. Figure 19 shows how the document is structured. A series of seven interactive system analysis activities feed the development of the appended data explained in subordinate paragraphs. The appended data then becomes the basis for lower tier analysis that produces content for the lower tier specifications and adds to the appended data.
2.1.12.1 Document Main Body

The main body simply contains a table of contents, list of illustrations, and list of tables for the document plus it should provide text explaining the capture of work products in the seven work areas during system and lower tier analyses. The body should also explain that the SDD couples the structured analysis work and its work products to specification content as guided by the selected specification standard templates.

2.1.12.2 Appendix A, Functional Analysis

This appendix captures the functional flow diagram starting with the identification of the system need and the life cycle flow diagram. The Use System Function is initially decomposed progressively to expose more details about the user need. For each block in the functional flow diagram, there should be one line in the function dictionary also contained in Appendix A.

2.1.12.3 Appendix B, System Environment Analysis

The environment consists of several subsets of stresses that are applied to the system. This appendix identifies and characterizes them. Timelines capture critical timing requirements. The spaces within which elements of the system must function are identified and the corresponding environmental stresses defined in terms of standards that describe those spaces. Service use profile analysis is applied to uncover end item environmental requirements. Finally, zoning of end items exposes component environmental requirements.
The system product entities result from the allocation of functionality to things. As these pairs are defined on the function-product entity matrix, they must be entered into the product entity structure diagram. This work should be accomplished by a team of people knowledgeable in system, hardware, and software engineering, manufacturing engineering, verification engineering, logistics, material and procurement, and logistics in order to evolve an optimum product entity structure which will be universally respected on the program. This product entity structure is also the basis for the specification tree. Each item on the tree must have a responsible agent identified, a template selected, and a release date established. This structure should also be the basis for any IPPT established on the program. It is also the basis for the WBS so the SOW and IMP align perfectly with the teaming structure applied on the program.

Interfaces are identified by pair-wise evaluation of function allocations to product entities using an n-square diagram. This appendix identifies all interface needs internal to the system as well as externally to the cooperative systems identified in Appendix B.

Appendix E provides a space in which system engineers can capture their work directed at identifying the specialty engineering disciplines that will have to accomplish work on the various entities in the system product entity structure to define the appropriate requirements and subsequently the needed analyses to confirm that those requirements are being satisfied. A specialty engineering scoping matrix is used to report the results of that analysis.

Appendix F captures the results of a physical process analysis in the form of a process flow diagram. This is used by logistics engineers to drive out requirements related to training, support equipment, maintenance procedures (tech data content), and spares consumption. It is also needed to complete the environmental use profile study reported upon in Appendix B that drives environmental requirements for end items.

The exposed functions are listed in the Requirements Analysis Sheet (RAS) contained in this appendix. Related performance requirements are defined and allocated to a product entity. These performance requirements have to have a paragraph number assigned, title identified, and they can be outputted into a specification following a particular template. That part of the work can be done inside a requirements database system. Ideally, all of this work would take place within a requirements database tool but some organizations may find it preferable for their purposes to do the traditional structured analysis work using pencil and paper followed by capture of the resulting requirements in a word processor or a computer database tool from which specifications can be generated.
In keeping with the Integrated RAS idea advanced in Paragraph 2.1.10, the RAS is not restricted to performance requirements. Specialty engineering, interface, and environmental requirements can also be included so that every requirement appearing in every specification on a program will transition from the analytical model from which it was derived into a specification via the requirements analysis sheet.

2.1.13 Team Activity During Requirements Work

The PIT is responsible for accomplishing all requirements analysis work focused on developing the system specification and the immediately subordinate specifications that will be the top-level specifications for the top level IPPT. In general, this analysis work will be accomplished using traditional structured analysis following the pattern described in this section. PIT initiates the analysis capturing the work products in the SDD thus initiating that document. Requirements derived from the modeling work are entered into the database application initiating the requirements capture. Any customer requirements documentation is also entered into the database and traceability established between these requirements and those developed from modeling efforts that appear in the system specification. Traceability is continued down to the subordinate specifications to be handed over to the top-level teams when formed.

The PIT identifies all system external interfaces and defines them in the system specification or the beginning of an interface control document. Internal interfaces are also identified and defined for the first tier entities below the system level and the next lower tier in order to complete the internal interface definition for the first tier. All requirements are entered into the requirements database application and traceability entered.

The PIT develops a specialty engineering scoping matrix and maps the needed disciplines to the product entities identified and coordinates the indicated domain experts to derive requirements for the system and top-level end item specifications. The system level environmental requirements are derived from system spaces identified and mapped to corresponding tailored standards.

With the top-level specifications completed, the PIT can bring the top-level IPPT aboard. As those teams assemble and become familiar with their specification and the program planning data prepared by the proposal team, they continue the requirements analysis process relative to the functionality of their product entity. The primary role of the PIT switches to integration and optimization across the IPPT. The teams enter requirements data into the requirements database and maintain traceability. As this process continues, it may become apparent that lower tier teams are required in which case the parent team takes over system responsibilities for them. The parent team in all cases must develop the top-level specification for any new team. Also, at some point, a team will identify an allocation of functionality to computer software and the continued analysis of that entity should switch to UML.
2.2 UML

2.2.1 Entry Analysis and Overview

While it is encouraged that the enterprise apply TSA today as the entry modeling technique, it is entirely possible to initially enter the problem space analysis for a system that will be implemented primarily in software with UML rather than TSA or SysML. The suggested process starts at the top with the problem expressed in the system need and illustrated in a context diagram, borrowed from modern structured analysis, like that shown in Figure 20. The context diagram expresses relationships between the system, represented by the bubble, and terminators, representing outside entities deriving benefits from the system and supplying things needed by the system to function, generally information in a computer software system but more general in a system that will be implemented using a collection of technologies. Figure 20 is an alternative to the traditional depiction of the general system as a block labeled System interacting with a block named Environment.

![Figure 20 Context Diagram](image)

While UML could be the entry modeling approach at the system level following the approach discussed in this section, the discussion to follow is based on the assumption that the problem space will be entered using TSA with a recognition at some point of a need to switch to UML as entities are identified that must be developed in software. Figure 21 illustrates a process for applying UML starting at whatever level in the system the program chooses to start applying it. Generally, this will be some level below the system level based on allocation of higher tier functionality to a software entity.

In a given system, the initial analysis may have identified one or more entities that will be implemented in computer software so for each of these separate entities one should build a context diagram. It should be noted that the context diagram was popularized in modern structured analysis and was not adopted by UML but it is a useful artifact with which to identify the inside-outside relationship between the software entity and the entities external to that software with which it must interact. The context diagram is offered as an intermediary view that will lend some discipline and order to the identification of use cases. We will identify one or more use cases for each terminator and perform a dynamic analysis on each of those use cases.
For each terminator in Figure 20, identify one or more use cases through which the intended functionality will be accomplished identifying the actors deriving benefits from the system to be created. One use case may expand into several extended or included use cases to cover a more complex situation. You will note that the opening gambit has been arranged to provide structure in the identification of needed use cases. Next, for each use case, build one or more scenarios dealing with how that use case interacts with the system. These scenarios can be in text form, a list of events, or some kind of diagrammatic treatment.

Then, express each scenario in the form of an activity and/or sequence diagram cast at the UML entry level initially. The activity diagram can be thought of a functional flow diagram similar to that used by software programmers many years ago or by system engineer in traditional structured analysis. It may be drawn vertically or horizontally as far as the author is concerned. Swim lanes may be overlaid upon the activity diagram each of which corresponds with the lower tier entities (classifiers in UML) that will be responsible for implementing activities within their swim lane. This is a key point in the analysis where the analyst must make lower tier product entity structure decisions that should lead to adding software entities to the product entity structure. Some analysts prefer to think of the two-dimensional artifacts in the diagram as states rather than activities or functions. Simultaneity is not permitted in normal state diagramming but UML permits it in activity diagrams to cover decisions and branching in a way similar to that applied in flow charting.

Alternatively, the analyst can use a sequence diagram to open the exposure of the details of a use case scenario. The entities (classifiers) through which the functionality and behavior are explored are identified on what are called the life-lines drawn as dashed lines below selected physical or
logical classifiers. These lifelines correspond with the swim lanes on the matching activity diagram. Directed line segments connect these life-lines to show passage of messages and relationships between the classifiers. As in the activity diagramming, these life-line decisions may identify entities that already have been identified or they may involve entities not previously identified. In the latter case, the PIT must concur in the model extension and add the new entities to the product entity structure.

Each of these diagrams (activity and sequence) identify a lower tier (white box) view of what entities will be needed to accomplish the exposed functionality and behavior, what will have to happen in the system in order to achieve the intended goals of the use case signified in its name, and offer a detailed view of the order in which these things will occur.

The analyst can allocate top-level functions (activities) to specific entities and arrange the blocks of the activity diagram in corresponding swim lanes linked to these entities. These swim lanes will correspond to nodes, or even higher-level entities, at the system level but in any case we can refer to them in general as physical or logical classifiers within UML. If appropriate, analyze each of these classifiers from a dynamic perspective using some combination of sequence, communication, activity, and/or state diagram. The communication diagram shows the same information as a sequence diagram with an emphasis on the entities rather than the relationships between those entities. A state diagram is useful where there exists some finite number of possible conditions within which the entity can exist and there appear to be understandable rules for the entity to change from one condition (state) to another.

Identify requirements derived from these artifacts and capture them in a program-wide RAS linked to the modeling artifact (MID) that encouraged their identification and the physical or logical classifier, referred to more generally by a product entity ID (PID) in the RAS, that will be responsible for responding to that requirement and in the specification for which it should reside.

2.2.2 The Connection Between Modeling Artifacts, Specification Content, and Product Entities

Figure 22 suggests a hierarchical relationship between the elements of the UML analysis and offers a way of assigning MID. The capabilities in the specification format (template Paragraph 3.2) coordinate with terminators, use cases, extended use cases, and/or scenarios. Figure 22 only shows one terminator expanded but the intent is that for any top level software classifier AX (highest tier SW entities) entered into the product entity structure, one or more terminators would be identified and expanded as shown in the one case shown in Figure 22. The software top-level software classifier, AX, is, of course, a member of the product entity structure (PID) where X is a string of base sixty or decimal delimited base ten numerals. The suggested UML MID stream is identified first with a unique UXh MID (with \( h = e\{1, 2, 3\} \) in the example shown in Figure 22) for each terminator.

The terminator MID can be further decomposed using the MID UXhijk pattern composed from a set of use cases, possible extended use cases, and scenarios for each terminator h. These MID are the entries to place in the RAS for software requirements derived from these artifacts. In general, capabilities will be derived from these artifacts and the requirements subordinate to them will be derived form the dynamic modeling artifacts UXhijk1 through UXhijk4.
So finally, the requirements for each capability flow out of applying the subordinate UML dynamic modeling artifacts. As in TSA with lower tier product entities identified from lower tier function allocation, the lower tier software product entities flow from the sequence (life-lines) and/or activity (swim lanes) diagramming. This is a particularly satisfying coordination between lower tier entities being exposed through functional analysis in TSA and activity analysis in UML where both are using essentially the same model to identify lower tier entities.

We can continue to apply UML in the lower tiers as covered in Figure 21 treating each classifier as a system in accordance with the steps discussed above progressively identifying nodes, components (possibly in more than one layer), and classes. If the system level problem space was entered using UML as this process moves forward and downward, it will be decided that some classifiers will be developed as software and hardware entities, with the latter possibly splitting in lower tiers into hardware and software entities. The continuing analysis of hardware entities can be accomplished using traditional structured analysis or SysML model artifacts while the modeling of software entities continues primarily using UML model artifacts. In that it is
difficult for software to contain hardware, the use of TSA as the entry analysis probably makes more sense.

At the lower tier UML analyses where the physical and logical classifiers are classes and objects, the lines that flow between the classifiers on the corresponding sequence and communication diagrams coordinate with messages and influences applied to/from those classes in relation to external physical and logical classifiers (other classes generally at this level). This same effect is in operation whether the classifiers depicted are classes, components, or nodes. Each classifier must have identified for it one or more operations (services or functions) that it performs relative to an outside set of actors and one or more data entities it deals with internally to accomplish those operations. The data elements will flow into the classifier via the lines on the sequence and communications diagrams and data may be created or altered internally. Initially, the analyst may choose to first identify node, component, class responsibilities and subsequently as the analysis matures translate these responsibilities into operations and attributes.

In lower tier analyses, the assigned IPPT are responsible for identifying and defining needed interfaces below the level initially identified by the PIT. Where the two terminals of an interface touch only entities that are the responsibility of a single team, that team is clearly responsible for interface identification, definition, and integration. Where an IPPT is responsible for only one terminal of an interface, that team must cooperate with the team responsible for the other terminal to develop that interface. The integration agent in this case is the lowest common team. If there is only a single layer of teams under the PIT, the PIT is always the lowest common team. In general, it should be the team on the receiving end of an interface that first identifies the need for a new interface since interfaces should not be defined based on what is available but what is needed. If it is not obvious what team shall be responsible for the new interface, the PIT shall act as the integration agent until such time as the source is identified and then the lowest common team will take over that responsibility.

As IPPT identify lower tier entities during use case analysis, the PIT shall approve those additions and assemble them into the formal product entity structure. These entities may be physical or logical entities but eventually all of them must be identified as real product entities that will be developed. These final entities can be logical entities as in the case of computer software that will run on a particular hardware computer entity. Where it appears that lower tier entities will entail significant complexity, new IPPT may be formed by the PIT that will be subordinate to the appropriate existing IPPT. Those lower tier IPPT will take over the continuing analysis of use cases appropriate at that level and the immediately superior team will take on the role of the system engineer for the new team just as the PIT does for all top tier IPPT.

The use case analysis process employed by an IPPT or a collection of teams will necessarily be a collaborative process involving people from several different specialty disciplines. Each such team will have a leader whose responsibility it is to bring the analysis to a conclusion as scheduled. These teams will come into being, exist for a brief period of time, come to a conclusion, and pass from the scene with others replacing them. Once a use case analysis has been completed by a team, the work products will have been captured in modeling applications and integrated relative the existing work products. The modeling front will move down through the advancing product entity structure till the system is fully characterized.
Where personnel from other teams are required to accomplish work on another team’s use case analysis, the owning team will cover the manhours of all personnel working the use case. Where all team members are physically collocated in the same facility, they may be depended upon to interact well through a traditional meeting in the common facility. Where at least some of the people required on a use case analysis effort are not collocated, it will be necessary to extend the meeting place geographically through the use of a product such as webex where people from several different geographical locations can cooperate in the development of a set of information.

Leaders of use case analysis teams are responsible for the prompt completion of team activity with good results but in so doing they will be well served to identify the most effective collaborative engineer on the team to lead collaborative team activity in the form of meetings especially where those meetings entail the use of distance integration aids like webex. The collaborative discussions in meetings need not necessarily be led by the team leader.

As the team completes its modeling and requirements work, the results should be reviewed by the parent team and, if approved, the team should be empowered to proceed with design at a level appropriate to the team responsibilities. The team must continue any responsibilities it may have for integration and optimization and leadership of subordinate teams that may still be involved in modeling work. This design work will entail some combination of hardware and software design development.

2.2.3 Dynamic Modeling Artifacts Explained

The use case diagram is considered a dynamic modeling artifact also but it is treated here as a transition medium between classifiers and the dynamic model set. The remaining dynamic models are implemented in four diagrams from which we may select any subset including all of them, any one of them, any pair, or any trio for a particular scenario analysis. It is not necessary to use them all for any particular analysis. Use those that make it possible to understand the problem space and properly characterize it in requirements. Some very large programs have done much of the analysis with only use case and sequence diagrams. The more complex the problem space and the more intimately the parts of the evolving system interact, the more different views of problem space that will be useful.

The first two kinds of diagrams covered, sequence and communications are both modeling the same relationships and collectively referred to as interaction diagrams. The sequence diagram emphasizes the time ordering of messages and the communication diagram emphasizes the organization of the objects that participate in the interaction. The second two are forms of state diagrams in the mind of many analysts.

2.2.3.1 Sequence Diagram

Some programs apply only the sequence diagram to explore the dynamic behavior of use cases and this may be adequate on relatively simple problems. In this tutorial we are assuming that the analyst employs the sequence diagram as the initial dynamic model, though an alternate approach would be to use the activity diagram for that purpose, but uses at least some of the
other three models to explore the use case more thoroughly. The sequence diagram example in Figure 23 illustrates the fundamentals showing two classifiers that comprise the classifier AX that has previously been analyzed. Here we conclude that AX must consist of AX1 and AX2 and that in order to accomplish the behavior defined for AX these lower tier classifiers must interact with an outside entity called an actor which will derive some kind of benefits from the relationship. The two subordinate classifiers will provide certain operations that are not clearly defined on this diagram. In the process of doing so, they will exchange messages in a certain order with time running down the page.

Each classifier including the actor has a lifeline in the form of a dashed line running down the page. A block is overlaid on this dashed line to indicate the active period(s) of classifier. Between these classifier lifelines we see messages being passed from one classifier to another. The names of these classifiers are formed of one or more words concatenated together without spaces and all but the first word capitalized. After the message name one can insert an argument list parenthetically.

The model permits the creation of classifiers and when they have performed their activity they can be killed. These features are not illustrated in Figure 23. The kinds of messages identified are: (1) a call invokes an operation on a classifier on the arrow end of the message, (2) a return message returns a value to the caller (dashed line used), (3) a send message sends a signal to a classifier, (4) a create message creates a classifier, and (5) a destroy message destroys a classifier. Message two and four could be an example of messages types 1 and 2. Message three is an example of message type 1 where the classifier is making the call upon itself.

![Figure 23 Sequence Diagram Example](image)

After a classifier sends a signal to another classifier the sending classifier returns continues its own execution. The target classifier independently decides what to do about it. A common reaction would be to trigger a state machine causing the target classifier to execute actions and change state.
2.2.3.2 Communication Diagram

In some cases we are primarily interested in the time ordered sequence of messages between classifiers but in other situations we may be more interested in the organization of the classifiers and a communication diagram can offer better results even though the sequence and communication diagrams are semantically equivalent. Figure 24 illustrates a communication diagram reflecting the same situation as Figure 23. The classifiers are joined by lines corresponding to the relationships between them and messages are related to these line each in terms of message name, direction, and the relative timing of the message.

![Communication Diagram Example](image)

2.2.3.3 Activity Diagram

An activity diagram can be used to express the things that one or more classifiers must accomplish. It is weak in terms of absolute timing of accomplishing those things but strong in expressing the relative order of those things. As shown in Figure 25, activities are illustrated by rounded corner boxes and they are connected into a sequence by directed line segments. In addition to the activities, we also will wish to show simultaneity through the use of forking and joining structures and alternative paths using branch and merge structures. The guard expressions give information about the conditions that correspond to movement through one branch or another.

This is the functional flow diagram of UML though many analysts prefer to think of the blocks as states rather than functions. The author prefers not to consider them states because it is in conflict with the notion that an entity must be in only one state at a time.

The diagram can be built with swim lanes, or not, that relate to classifiers, the same classifiers identified on sequence diagrams using the lifelines. It is through these two devices that we can allocate software functionality to classifiers. As we do so we determine the next lower tier product entity structure and should offer up newly identified entities to the PIT for inclusion in the product entity structure.
2.2.3.4 State Diagram

An interaction diagram (sequence or communication) models the relationships between a collection of classifiers while a state machine models the behavior of a single classifier. The classifier in this case can be the whole system or a classifier at any level of abstraction. The state machine models the possible condition that a classifier can exist in and the transition of that classifier from one condition or state to another based on a stimulus that might be a signal from another classifier, the passage of time, receipt of a call message invoking an operation, or a change in some condition.

A state is drawn on a state diagram, as shown in Figure 26 showing a state machine for temperature control, as a rounded corner box with the name of the state written inside. Generally the diagram must have an entry and final state symbols though it is possible that once entered the state machine may continue forever. In some cases the intent may be for the machine to continue forever, as in a traffic light system, the reality is that such a system may have to be shut down for maintenance and does need a final state. Transitions between states triggered by events in the life of the classifier being modeled are shown diagrammatically as directed line segments between a pair of state and named in a way to convey how that transition is triggered. It is understood that the machine must be in only one state at a time and that only a single transition is possible at one time. Transitions are generally thought of as taking place instantaneously.
While UML does not prescribe it, a pair of dictionaries can be helpful in clearly stating the intended operation of a state diagram. One dictionary lists the states and defines them with precision while the other lists and defines the transitions. You will note that the transitions in Figure 26 have not been named uniquely but should be in the general case so that each can one can differentiate between them in such a listing.

2.2.4 Static Entity Analysis

In early object oriented analysis (OOA) as prescribed by Grady Booch, Peter Coad teamed with Edward Yourdon, James Rumbaugh, et. al., and many other practitioners, the proper way to enter problem space was using the static view with objects. Then they encouraged the analysis of the objects from a dynamic perspective with data flow diagrams for functionality and state diagrams for behavior. This approach is possible with UML using the foursome of dynamic modeling artifacts discussed in the prior section and it can be effective when developing a largely precedented system that can be observed in the real World like a new payroll system. The author believes that largely unprecedented problems are best attacked using Sullivan's encouragement of form follows function leading with the dynamics views and identifying the static entities that populate the product entity structure from a software perspective.

UML identifies three levels of static entities but they are all product entities and while drawn on modeling diagrams using different images, they are essentially the same at different levels of abstraction. All are simply illustrated on the system product entity diagram illustrated in Figure 14 as blocks. Figure 27 illustrates the three static entities collectively referred to as classifiers in this tutorial.
In this tutorial the case is made for first identifying the nodes which are entities that will be associated with run time software. They are higher-tier entities. Like classes and components, they have associated attributes and operations. They interface with each other and possess lower tier interfaces between components and classes that comprise them. The nodes are identified in the dynamic analysis of the top-level software entity by identifying sequence diagram lifelines and activity diagram swim lanes. We then analyze these nodes from a dynamic perspective and identify components in the same fashion.

The alternative approach first identifies classes corresponding to observable entities in the problem space which are then dynamically analyzed leading to an understanding about how best to package these entities based on collecting the classes with the most intense interface relationships together as components. Just as in the use of interface analysis in TSA to validate the product entity decisions in functional analysis by observing possible unintended interface intensities, we can in UML re-consider the particular swim lanes and lifelines we selected in the dynamic analysis.

In this section, the intent is to explain what the UML static entities are and how they are used on diagrams. We will use classes in order to do so with the understanding that nodes and components are but higher tier classes. First we will describe a general class then explore structural relationships between these classes and finally we will cover the messages that are passed between them.

2.2.4.1 The Class

A class is illustrated as a box as shown in Figure 27c. The name of the class is placed in the top portion of the box, attributes are listed in the middle portion of the box, and operations are listed in the lower portion of the box. A forth box can be included below the operations in which we inscribe class responsibilities in free-form text. A responsibility is a contract or an obligation of a class. You may find it useful to begin the analysis of classes this way translating these into attributes and operations that best fulfill the class’s responsibility as the model is refined.
2.2.4.2 Class Relationships

Figure 28 illustrates the structural relationships recognized between classes. A class is said to be dependent on another if it depends on that other class for information or services. A class can be linked hierarchically to another through a generalization. Class associations can have the three adornments illustrated in Figure 29.

NameOne is the base class. NameTwo and NameThree are leaf classes in a generalization.

NameOne depending on NameFour for information and services.

An association is a structural relationship

- **Association Name**
  
  NameOne \( x \cdots y \) NameTwo

- **Association Role**
  
  The face that the class at the far end of an association presents to the class at near end of the association. Role names called end names.

- **Association Multiplicity**
  
  Tells how many objects may be connected across an association instance. Given by a range of numbers.

- **Association Aggregation**
  
  Expresses a whole-part relationship between to associated classes.

Figure 28  Structural Relationships

Figure 29  Association Adornments
2.2.4.3 Messages

Classes can also be related through the five kinds of messages discussed in Paragraph 2.2.3.1. Such a message can convey to a class a variable argument (value) that is needed in a class operation or it can convey a signal that causes the class state machine to transition to a new state for example. The messages that must be passed between classes are understood in the context of the sequence diagram under the assumption that the dynamic analysis is accomplished prior to the static analysis.

2.2.5 Related Analyses

2.2.5.1 Specialty Engineering

The specialty engineering matrix discussed in paragraph 2.1.9 can be used in software as well as hardware to identify all product entities for which specialty engineering requirements must be developed. This includes, for example, safety, reliability, and security. The software interface requirements fall out of the sequence and communication diagram analyses and flow into the specification template offered in Figure 27.

2.2.5.2 Environmental Requirements

Software environmental requirements are somewhat different from the hardware and system environmental requirements that tend to be dominated by the natural environmental factors. The software being an intellectual entity rather than a physical one, is shielded from the natural environmental stresses. True, the software operating within a machine that can suffer adverse environmental stresses can as a result fail, but this is a secondary effect. Reasonable software environmental relationships include any language restrictions and the computer architecture upon which it must run, for example.

2.2.6 Specification Structure

The specification outline offered in Figure 5 can also be applied to software entities where the capabilities are linked to either the terminators, use cases, extended use cases, or scenarios and the subordinate requirements listed under each capability are drawn from the corresponding dynamic diagramming (activity, sequence, communication, and state diagramming) work. Thus the requirements can be clearly shown to trace to modeling artifacts just as the performance requirements in hardware can be shown to flow so clearly from functions. Figure 30 offers an outline for a software requirements specification (SRS) using an edited template from EIA J STD 016 to integrate the supporting modeling work into the specification. Note the similarity to the outline in Figure 5 for a system or hardware item performance specification.
<table>
<thead>
<tr>
<th>PARA</th>
<th>TITLE</th>
<th>MODEL</th>
<th>DEPT</th>
<th>APP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>SCOPE</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>2.</td>
<td>APPLICABLE DOCUMENTS</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.</td>
<td>REQUIREMENTS</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1</td>
<td>Required states and modes</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.1</td>
<td>Classifier context diagram</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.2</td>
<td>Use case analysis</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.2.h</td>
<td>Terminator h</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.2.h.i</td>
<td>Terminator h, use case i</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.2.h.i.j</td>
<td>Terminator h, use case i, extended use case j</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.3</td>
<td>Dynamic Analysis</td>
<td>DID</td>
<td>2100</td>
<td>D</td>
</tr>
<tr>
<td>3.1.3.h</td>
<td>Terminator h dynamic analysis</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.3.h.i</td>
<td>Use case hi dynamic analysis</td>
<td>UML</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.3.h.i.j</td>
<td>Extended use case hij dynamic analysis</td>
<td>UML</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.3.h.i.j.k</td>
<td>Scenario hij dynamic analysis</td>
<td>UML</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.3.h.i.j.k.1</td>
<td>Sequence diagram hijk1 analysis</td>
<td>UML</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.3.h.i.j.k.2</td>
<td>Communication diagram hijk2 analysis</td>
<td>UML</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.3.h.i.j.k.3</td>
<td>Activity diagram hijk3 analysis</td>
<td>UML</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.3.h.i.j.k.4</td>
<td>State diagram hijk4 analysis</td>
<td>UML</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.4</td>
<td>Lower tier classifier identification</td>
<td>UML</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.2</td>
<td>Entity capability requirements</td>
<td>UML</td>
<td>2100</td>
<td>A</td>
</tr>
<tr>
<td>3.2.m</td>
<td>Capability m</td>
<td>UML</td>
<td>2100</td>
<td>A</td>
</tr>
<tr>
<td>3.2.m.n</td>
<td>Capability m, requirement n</td>
<td>UML</td>
<td>2100</td>
<td>A</td>
</tr>
<tr>
<td>3.3</td>
<td>Interface requirements</td>
<td>UML</td>
<td>2100</td>
<td>D</td>
</tr>
<tr>
<td>3.3.1</td>
<td>External interface requirements</td>
<td>UML</td>
<td>2100</td>
<td>D</td>
</tr>
<tr>
<td>3.3.1.m</td>
<td>External interface m</td>
<td>UML</td>
<td>2100</td>
<td>D</td>
</tr>
<tr>
<td>3.3.1.m.n</td>
<td>External interface m, requirement n</td>
<td>UML</td>
<td>2100</td>
<td>D</td>
</tr>
<tr>
<td>3.3.2</td>
<td>Internal interface requirements</td>
<td>UML</td>
<td>2100</td>
<td>D</td>
</tr>
<tr>
<td>3.3.2.m</td>
<td>Internal interface m</td>
<td>UML</td>
<td>2100</td>
<td>D</td>
</tr>
<tr>
<td>3.3.2.m.n</td>
<td>Internal Interface m, requirement n</td>
<td>UML</td>
<td>2100</td>
<td>D</td>
</tr>
<tr>
<td>3.4</td>
<td>Specialty engineering requirements</td>
<td>DID</td>
<td>2100</td>
<td>E</td>
</tr>
<tr>
<td>3.4.m</td>
<td>Specialty Engineering Discipline m</td>
<td>Specialty Scoping</td>
<td>E</td>
<td></td>
</tr>
<tr>
<td>3.4.m.n</td>
<td>Specialty Engineering Discipline m, Requirement n</td>
<td>Specialty Scoping</td>
<td>E</td>
<td></td>
</tr>
<tr>
<td>3.5</td>
<td>Environmental conditions</td>
<td>3-Layered Env Model</td>
<td>2100</td>
<td>B</td>
</tr>
<tr>
<td>3.6</td>
<td>Precedence and criticality of requirements</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>4.</td>
<td>VERIFICATION</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>5.</td>
<td>PACKAGING</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>6.</td>
<td>NOTES</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>6.1</td>
<td>Requirements traceability</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>6.1.1</td>
<td>Inter-specification traceability</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>6.1.2</td>
<td>Verification traceability</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>6.1.3</td>
<td>Modeling traceability</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>6.1.4</td>
<td>Section 2 traceability</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>6.1.5</td>
<td>Programmatic traceability</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>6.2</td>
<td>Glossary</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>6.3</td>
<td>Specification maturity tracking</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
</tbody>
</table>

Figure 30  Software Requirements Specification (SRS) Template
Refer to Exhibit B of this student manual for a JOG System Engineering data item description (DID) that shows how the subparagraphs of paragraph 3.1 relate to the subparagraphs of paragraph 3.2. Paragraph 3.1 essentially provides an opportunity to capture the complete UML analysis for the classifier covered in the specification. This can be done by actually including the diagrams discussed in prior paragraphs in this text or by referencing them in a computer application within which the modeling is done and derived requirements captured or the diagrams can be completed manually (or with a computer graphics application like Microsoft Visio or Powerpoint) and contained in the appendices of the system definition document (SDD).

The reader will note that in Figure 19 there is a second plane labeled UML for the case where the program chooses to capture the UML analysis work products in the SDD. On a program that is dealing only with a UML analysis for a software product the Appendix structure shown in Table 1 column A could be used. If the program must deal with both TSA and UML, the software appendices could be simply added to those required for TSA and lettered G through N with the TSA Appendix G (the RAS that should also contain the software requirements derived from UML artifacts using the MID pattern illustrated in Figure 22) becoming a common RAS in Appendix O. The latter pattern is captured in column B. In this combined case, the classifier diagramming can remain in Appendix N but Appendix C should capture the aggregate product entity structure including hardware and software entities.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>MODEL ARTIFACT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>G</td>
<td>Context Diagrams</td>
</tr>
<tr>
<td>B</td>
<td>H</td>
<td>Use Case Diagrams</td>
</tr>
<tr>
<td>C</td>
<td>I</td>
<td>Scenarios Text</td>
</tr>
<tr>
<td>D</td>
<td>J</td>
<td>Sequence Diagrams</td>
</tr>
<tr>
<td>E</td>
<td>K</td>
<td>Communication Diagrams</td>
</tr>
<tr>
<td>F</td>
<td>L</td>
<td>Activity Diagrams</td>
</tr>
<tr>
<td>G</td>
<td>M</td>
<td>State Diagrams</td>
</tr>
<tr>
<td>H</td>
<td>N</td>
<td>Classifier (Object, Class, Component, and Deployment) Diagrams</td>
</tr>
<tr>
<td>I</td>
<td>O</td>
<td>RAS</td>
</tr>
</tbody>
</table>

The reader should note that in software using UML it is possible to relate the modeling machinery very clearly with the classifiers about which specifications are to be written. It is not quite so easy in hardware using TSA because all of the performance requirements derived from a particular function may be allocated to the same entity. It might be possible to force fit the functions into a closer alignment with the product entity structure by restarting the functional analysis for each entity identified but the author does not think this would be particularly helpful.

2.2.7 Software Requirements Close-Out

When a set of classes that collectively form a component have been thoroughly characterized including the sets of requirements associated with those classes and dynamic relationships and the related requirements have been captured, reviewed, and approved, the responsible team can
start to develop the computer code that will implement the component. This process can begin across the lower tier of the components and continue to next assemble into computer code for the nodes. This process may take place in a single string or simultaneously following the same pattern in multiple paths where the software is distributed to some extent. There is, of course, a continuing need for integration and optimization across the development being accomplished by different teams that focus primarily on interfaces just as in hardware development.

The PIT and program manager will reach a conclusion for the program with possible local differences whether to pursue software design and coding in a top-down or bottom-up fashion. In either case, any code developed must be tested at each level of build. This may require the development of special stub (top-down) and/or drivers (bottom-up) to permit testing of completed portions before higher or lower tier software has been completed.

The classifiers identified in the UML analysis should be entered into the product entity structure by the PIT so that a consolidated view of the overall system matures. Figure 31 shows the product entity structure evolving based on feed from all modeling activities being applied on the program under the watchful integrating eye of the PIT.

![Figure 31 Evolving Product Entity Structure](image)

The UML analysis process encouraged does involve a partitioning of the whole problem space into parts hierarchically arranged and assigned to teams. These teams, in hardware and software, can generally be depended upon to integrate and optimize at the level for which they are responsible rather than the next higher level. Therefore the PIT (and superior IPPT) must integrate and optimize across the best efforts of the teams. The PIT should also consider special integration studies across the grain of the system based on overarching functions such as shutdown and activation as well as others that may be suggested through system level states and modes analysis. These efforts should be led by the PIT and participated in by people from a wide range of teams.

### 2.3 Opening the Analysis With DoDAF

DoDAF was developed to support modeling of the complex information systems that DoD has been assembling in recent years to interconnect their many sensor and weapons systems to form
an effective military capability often referred to as sensor-to-shooter connectivity. It does not provide a complete modeling set primarily because of the absence of a model artifact for the physical product entities. The product entity view can be provided by augmenting the DoDAF model set with the TSA product entity diagram, however, from the aggregate model set this description assembles. Physical products identified in the DoDAF process can be inserted into the TSA product entity diagram just as they can be using any of the model sets and as suggested in Figure 31.

The DoDAF problem space entry is similar to the UML entry and in fact can use some UML artifacts to do so. DoD does not prescribe any particular set of modeling artifacts for the several views. In this paper we will use a combination of UML, IDEF, and TSA artifacts. DoDAF employs four modeling views: (1) two all view products that offer a textual overview of the system (as in what was once called a mission need statement) and an overall glossary explaining all terms used; (2) two technical view products that define standards that must be respected on the program and the evolution of those standards over time; (3) nine operational view products that capture how the user views their needs; and (4) thirteen system view products that offer the engineering perspective on the needed system. Figure 32 illustrates a recommended DoDAF development sequence. The simulation work could better be shown overarching both the analysis and design work interacting with and serving both. Note that the two all and two technical views would be a good addition to any modeling apparatus.

![Figure 32 DoDAF Development Sequence](image)

The technical and all views work products should be developed initially by the user and acquisition agents and evolved by the contractors selected to develop the system. The user should have developed the operational views in the process of evolving a set of documents used to refine their need. These documents include a Joint Capabilities Document (JCD) that takes the
place of the old need statement cut very broadly. In the study process prescribed by DoD for a new system, the capability identified in the JCD may be cut up into n systems each of which should have developed for it an Initial Capabilities Document (ICD) that is the equivalent of the old mission needs statement for a given system to be procured. The ICD is then matured into a Capabilities Development Document (CDD) and eventually a Capabilities Production Document (CPD). In between these last two a contract would commonly be let resulting in the development of a system specification attached to the contract. DoDAF can be used to gain a systematic insight into the content needed in all of these documents.

2.4 Integrated Modeling

A program must make a decision about the modeling techniques it will apply as it builds the proposal. This may include some mix of TSA, UML, SysML, DoDAF, and IDEF-0. A program that is going to be primarily computer software or networked assets could enter the program with UML or DoDAF. If the product is a database, it could enter with IDEF-0 or UML. But, generally, modeling entry should involve the use of TSA or SysML at the system level because software, an intellectual entity, must run on hardware entities that provide the product with real substance. As noted earlier, at the time this was written SysML was not yet fully ready for use so TSA would be the author's preference now. But, an enterprise should continue to follow the development of SysML and work toward replacing TSA with SysML. In any case, there is a need to recognize that for some period of time there will be a need for a model that works well for systems and hardware and another model that works well for SW. For now, also, there is not one great computer application within which one can model HW and SW requirements work and permit easy cross model traceability and provide specification publication capability so it will be necessary to use two or three applications to cover the needed tool set.

Work can be accomplished well for systems and HW as covered under paragraph 2.1 and for SW using the approach covered under paragraph 2.2. When applying both, problems will tend to arise when transitions have to be made between these two approaches. It is not possible for SW to include HW but the opposite case is perfectly normal. So, the transitions will only be a problem as the analysis shifts from HW to SW moving from the use of TSA or SysML to UML. There are two concerns at this point: one in the models applied and the other in the computer applications employed.

The transition point will occur when the highest tier software entities are identified. There may, of course, be several of these transitions distributed about the expanding product entity structure. The program has the option of pooling all of the software into an integrated entity or permitting it to be distributed within multiple processors that may still all be under the responsibility of one team or distributed among teams with both hardware and software responsibilities. If we can solve one of these hardware-software handoffs we will have solved the general problem of requirements traceability across these gaps.

It should be clear that requirements traceability to models is assured in the approach covered in this tutorial because all of the requirements are to be derived from a model. Vertical or hierarchical requirements traceability is very simple in specialty engineering areas in hardware, software, and across the gap as described in this tutorial. The environmental requirements are
vitaly different between hardware and software and one can make a case that lower tier software environmental requirements should not have to respect traceability across the gap to higher tier hardware or system environmental requirements that are largely environmentally related. Precisely the same method of identifying hardware interface requirements can be used to identify software interfaces as well as hardware-software interfaces because we identify them between entities that appear in the joint product entity structure. So, if interface requirements traceability involves lower tier interface expansion requirements to higher tier interface requirements, traceability is assured. This leaves only the performance requirements a remaining problem from a vertical or hierarchical traceability perspective.

Given that the system entry analysis was accomplished in TSA using some form of functional analysis and the lower tier software analysis is going to be done using UML, there is a temptation to employ activity diagrams in UML to analyze software entities from a dynamic perspective because it is very similar to functional flow diagramming and might give us some interesting opportunities to link up hierarchical traceability. However, for a given software entity there may have been 10 performance requirements derived from 8 functions allocated to the software entity in question. There is no clear way to link up the activity analysis and requirements derived from it with the several functional analysis strings and the performance requirements derived from them that can easily be automated.

So, let us pursue another tack in an attempt to coordinate the traceability relative to the sequence oriented dynamic analysis approach described previously. If requirement R@1 is one of a set of requirements R%1 through R%10 where R%1 is derived from function F#1 of a set of functions F#1 through F#8 and requirement R%1 is allocated to product entity AX2 and it is decided that AX is going to be developed as a software entity, then one of the scenarios to be analyzed will be UXhijk1 from which we derive requirement R@1. What we are looking for is a way to establish hierarchical traceability between requirement R@1 and some requirement in the set R%1 through R%10. The X, %, and # characters are being used to designate base 60 strings in this discussion. We know that requirement R@1 must be traceable to one of the 10 performance requirements allocated to classifier AX2 and we can look at that list of requirements and select the one most closely related.

To make this selection more organized, we can form an x by y matrix, in this case a 10 by 12 matrix, and pair-wise compare the sets R@ and R%. In Figure 33 you can see this whole process taking place The 10 functionally derived requirements are captured in the RAS mapped to the set of functions F#1 through F#8 and allocated to product entity A&. Based on these requirements we build a context diagram for entity A& and analyze A& from the perspective of each of the three terminators shown. As an example Use Case U&3 is extended to three use cases and we build three scenarios one of which, U&3111 is analyzed from a dynamic perspective with some combination of sequence, communication, activity, and state diagrams. Requirements R@1 through R@12 are derived from these analyses and captured in the RAS (possibly linked to the RAS database from a UML modeling application).

There are 10 requirements (R%1 through R%10) to which the requirements in the set R@1 through R@12 will have to hierarchically trace. We can build a 10 x 12 matrix and pair-wise analyze the
relationships between the two sets of requirements, perhaps concluding that one of the matches is \( R@_1 \) traces to \( R%_1 \). All of the matches are marked in the requirements management database table for traceability relationships. There are no known databases that provide the traceability evaluation matrix so it may have to be accomplished as a pencil and paper aid. However we should be able to set the requirements management database filter for the two sets of requirements of interest aiding in the identification of the sets of interest for a particular case. In this example, there might be ten or more sets of requirements like \( R@ \) derived from ten or more dynamic analyses. In each case an \( x \) by \( y \) matrix would be needed to pair-wise analyze the traceability relationships. In any case, it should be clear that we can have good requirements to modeling traceability and even good hierarchical traceability across the gap between performance requirements.

![Traceability Evaluation Matrix](image)

**Figure 33  Requirements Traceability Across the Gap**

In both TSA and UML we have discussed a decomposition process that partitions the problem space into parts in which the analysis is accomplished. Whenever we partition any whole we have an obligation to integrate and optimize across the boundary conditions thus created. The PIT must accomplish this integration work relative to the top level IPPT and each IPPT with lower tier teams must accomplish this work relative to it's own immediately subordinate teams. Much of this integration work will take place at the interfaces ensuring that requirements on one end of an interface are compatible with those for the other terminal. Each team with subordinate teams, however, should also integrate across its immediately subordinate teams relative to the requirements derived at the subordinate team level relative to those at the parent team level. Part of this work can be accomplished by simply establishing the traceability between the requirements at the two levels. Another approach of value is to accomplish higher tier function effects across the lower tier team responsibilities. For example, one can inquire collaboratively into lower tier performance of higher tier functions like turning the system or entity on or off,
moving from one major mode to another, accomplishing some kind of transfer function, or physical separation or joining of two entities.

Another kind of traceability can also be used to stimulate integrating results. This was pointed out to the author by an engineer at Puget Sound Naval Base in Bremerton, WA. Given a requirement at level m, we can inquire if the intent of the requirement was fully implemented in the requirements for the n entities at level m+1 (downward). This kind of traceability inspection must await the development of the subordinate specifications, of course, as does all hierarchical traceability.

At one time, in the 1950s when software was a very young discipline, it happened that hardware and software analysis, to the extent that it was done, was accomplished using exactly the same model, flow charting as shown in Figure 34. Over time, probably encouraged by the ease with which flow charts could be outputted onto line printers using ASCII symbols, computer software people got into the habit of building flow charts in the vertical rather than the horizontal axis still used by system engineering in their functional flow diagrams. The activity diagrams of UML still reflect the vertical orientation but it is really of little significance which orientation is used. The absolutely fascinating approaching reality is that system and software people will rejoin the same house in the near future. As UML and SysML become more fully integrated as suggested in Figure 34, we will achieve a tremendous milestone of universal unified modeling capability.

As we pass through this door into a world of integrated modeling and supporting computer applications, we will find it a more reasonably affordable task to integrate across the hardware-software boundary than has been the case for many years. But then as now, integration takes place in the minds of the system engineers working on the program. These engineers must be
every vigilant for inconsistencies between information sets that signal that two different domains are not working from a common understanding of the problem and solution spaces.

Figure 35  The Approaching Merge

3  Requirements Management

3.1  Summary of Team Activity During Requirements Work

During initial product development work, the PIT will model the problem space using a predefined set of modeling methods selected from the list of enterprise-approved modeling methods and apply those methods to identify top-level system entities and interface relationships and their requirements. This level of system definition shall be completed before program level IPPT are initiated on the program and staffed. These top-level product entities are the basis for assignment of these teams. When a team is established, the leader shall be presented with the specification for the top-level entity for which the team shall be responsible, a design concept (in particular if it is HW or SW), a clear definition of all external interfaces, and the corresponding components of the WBS, SOW, IMP, and IMS. The team will be charged with the development of that item and all subordinate entities and interface relationships. In the process of so doing, the team may conclude that lower tier teams are required and must request that action of the PIT.

The work products from the IPPT will be loaded into the computer applications used on the program by PIT and checked for consistency. The PIT shall perform integration and optimization work on modeling work contributed by the IPPT fitting the work products into a coherent system analysis of the problem space. As part of this work where the product is to be implemented in SW, the responsible team will seek to identify all needed use case analyses and assign them for completion by people on the team if possible. Where this work must involve people from other
teams, the responsible team must request help from the PIT and a cross team analysis effort will be established. The responsible lead team must ensure that each analysis is complete with all needed supporting modeling work.

The PIT shall maintain the product entity structure, the interface relationships, and all requirements modeling and management assets. The requirements shall be retained in a relational database from which specifications may be printed to paper or computer screen and within which traceability may be maintained. This database shall be linked to one or more modeling applications used on the program. The modeling applications shall be used for the purpose of identifying the system entities, interfaces, and appropriate requirements in each case. All content of these applications shall be under the control of the PIT until such time that it is formally approved at which time it shall fall under configuration management control and shall not be changed without a formal approval as well. Responsibility for data entry can be distributed to IPPT or retained by PIT. Entry may be aided by special Microsoft Office applications making it unnecessary for personnel to develop and maintain computer application skills.

The PIT will seek to establish IPPT overlaid upon the product entity structure so as to minimize the interface relationships between the entities for which the teams are responsible. The purpose of this arrangement is to minimize the need for cross team coordination and staffing for use cases analyses.

The preferred lower tier HW development approach entails a continued application of TSA using the same modeling database application applied at the system level. The preferred SW product development approach entails PIT and IPPT application of unified modeling language (UML). In the near term traditional structured analysis (TSA) will be applied in combination with UML maturing to a combination of UML and SysML as the latter matures into a formally released model by the Object Modeling Group. TSA or SysML are to be used initially to identify system and top level product entities that will more often be hardware end items. As the analysis proceeds downward and identifies a need for computer software, the analysis should switch to UML.

### 3.2 Requirements Tools Base

Figure 36 illustrates the preferred tools environment for programs. A requirements management database is used to capture all of the requirements that will be published in specifications and those specifications may be published from this database. In Figure 36 this is referred to as a big dumb database with no slur intended for the makers of tools that do not include integrated modeling capabilities. This is a relatively simple relational database application that can contain text information in a tabular structure. Each table row corresponds to a unique requirement with data captured in table columns for the several fields needed. Additional relational tables may be required for vertical traceability, verification traceability, and lateral (to models from which the requirements are derived) traceability. The program may use available modeling applications for UML (such as Rational Rose) and TSA (such as CORE) and arrange for traceability between these applications and the management application in an application like DOORS.
Many enterprises find it difficult for engineers to maintain currency with a set of requirements database applications as well as other applications more directly related to their work. All or most of these engineers will, as a function of accomplishing their normal work, maintain proficiency with the three fundamental Microsoft Office applications (Word, PowerPoint, and Excel). Loader applications crafted from Microsoft Office applications may be used to permit all engineers to enter data into the requirements database suite without a need for the engineers to become intimately familiar with these applications.

The PIT must exercise integration and optimization control over the requirements application suite and will require some members who really understand the applications, how they work, and how their content is inter-related. The suite must be set up to permit passing control of approved content to configuration management while retaining control of all in-work data.

3.3 Recommended Specification Responsibility Pattern

In the author's view, a program should staff a program integration team (PIT) that should begin the requirements analysis process at the system level and develop the top level diagrams in the SDD. This work should continue as necessary to develop the content of the system specification and the specifications corresponding to the top-level teams. The structured analysis for each of these teams should be taken over by the corresponding IPPTs in each case until they have completed the content of the specifications that define the problem for any subordinate teams. If no subordinate teams have been identified then they would have to complete the analysis needed to develop all of the specifications subordinate to their top-level specification. This same pattern
carries down to the lowest level. Each team should act as the system agent for all of its lower tier teams and principal engineers. This starts at the PIT for the system and works its way down through the lower tier teams. The Program Manager and Chief Engineer/PIT Manager should review and approve the system specification and all top-level specifications. PIT should establish rules for review and approval of lower tier specifications created by the teams.

With different parties doing the structured analysis, it is necessary to apply process integration and the PIT should do that accepting data into the several appendices of the SDD, numbering the figures, and cross-checking the data submitted. At least one team will be involved in software development and if traditional structured analysis has been applied for the system level, then that or those teams responsible for software will want to switch to some form of software modeling such as UML. Regardless of the modeling methods applied, all of the requirements modeling artifacts should be captured in the SDD either in the paper appendices noted earlier or referenced in the database systems used. The integrated RAS should be implemented in the big dumb database of Figure 36 as simply the basic relational database table used to capture requirements.

3.4 Requirements Risk Management

The principal risks that appear during the requirements development work involve a sensed inability to satisfy the requirements. The risk may be motivated by the conclusion that insufficient financial, or schedule resources have been made available. The concern may be that the requirement simply cannot be satisfied with available technology. Finally, the concern may be motivated by the conclusion that the value is simply too hard to achieve with available skill and knowledge. It is not uncommon to partition all into the categories of cost, schedule, technology, and performance as a result.

3.4.1 Requirements Validation

EIA 632 identifies an activity called requirements validation where we make an effort to determine to what extent we can satisfy particular requirements. The simplest way of reaching this conclusion is to simply ask the person(s) responsible for accomplishing the related design work if they can satisfy the requirements. If there is a lack of confidence then we need to proceed further in our efforts to identify potential performance risks. As requirements are identified and written we should validate them at that time. In most cases, the conclusion will be that there is no problem. Should we conclude that either there is a problem or we are not certain that can satisfy the requirement the first alternative investigated should be to ask if the requirement can be changed making it more certain that it can be satisfied. If that is not possible, then we should choose a means to mitigate the risk through an appropriate analysis, development evaluation test, simulation, or demonstration. If we believe that the requirement is very important in the development effort and that it will require some time to reach a final conclusion, we may select the requirement for more intense management as a technical performance measurement (TPM).

Parameters are managed through TPM by placing them on a list and assigning each TPM to a specific engineer who is charged with closing the gap between the required value and the currently demonstrated capability. The parameter principal engineer must maintain two charts: (1) a parameter chart that tracks these two values over time annotated with notes citing important
events coinciding with significant changes in the relationship between the values and (2) an action plan stating what is going to be done, when, and to what anticipated effect.

3.4.2 Margins and Budgets

Every program manager will experience difficult problems each requiring a tough decision. These problems can be made less severe by ensuring that the program manager has the resources available. This outcome is encouraged where the values for the most difficult requirements are combined with margins such that there is an opportunity to award engineers with a very difficult design problem some slack. These margins come in three varieties: cost, schedule, and performance. Cost margin is commonly applied as a management reserve such that the program manager can award a team more cost to solve a problem. Similarly, if a team has a design problem that can be solved through award of schedule slack time, the design may be possible. The third kind of slack is requirements margins. In all of these cases, the margin is derived by invariably making the job more difficult. Cost margin is often made available by skimming the task estimates of 10-15%. Schedule slack is obtained by subtracting available time for tasks on the critical path. Risky requirements are made more difficult to achieve. For example a weight margin may be realized by subtracting 5% from all weight figures. So the engineers will be challenged to accomplish their design with a weight value of required value - 5%. This is more difficult, clearly. The good news is that engineers will most often make these more demanding requirements preserving the margin values for the most difficult problems. When a very difficult weight problem appears, the manager can allocate some available margin. The margins invariably will be consumed before the design process is complete but there are ways of using available margins from requirements not of the same kind. For example, if an engineer has difficulty reaching his/her reliability figure after all of the reliability margin is gone, the manager can award some cost margin to permit the use of better parts or some mass margin permitting a heat sink that will reduce junction temperatures.

Requirements budgeting also has a risk reduction effect because it partitions available requirement values to the several designs at any one level of indenture. For example, given that it has been decided that 1500 watts of electrical power will be available from a source and there are 10 loads to be attached to this source, an engineer must partition this available power in a rational way between these several loads and integrate the results.

3.4.3 Risk Tracking

Risk is often measured using a dual axis criteria dealing with the probability that the concern will be realized and the degree of difficulty it will present if it does come to pass. This makes it a little difficult to track a single risk parameter over time and the way many people apply the dual axis system makes it difficult to accumulate a program metric that can be tracked over time. A variation on the safety hazard index described in MIL-STD-882 offers a way to measure risk with a single parameter that responds properly to characterize instantaneous values and a historical record for the program.

Figure 37 shows the risk matrix. Tables 2 and 3 corresponding provide the dictionaries explaining the values entered on the matrix axes. The intersections contain an index number that is simply the product of the axis numbers.
Table 2  Risk Probability of Occurrence Criteria

<table>
<thead>
<tr>
<th>CAT TITLE</th>
<th>P(O)</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nearly Certain</td>
<td>0.95-1.00</td>
<td>Will occur at least once during program</td>
</tr>
<tr>
<td>Probable</td>
<td>0.75-0.95</td>
<td>Will probably occur once during program</td>
</tr>
<tr>
<td>Possible</td>
<td>0.50-0.75</td>
<td>May occur during program</td>
</tr>
<tr>
<td>Unlikely</td>
<td>0.25-0.49</td>
<td>Will probably not occur during program</td>
</tr>
<tr>
<td>Nearly Impossible</td>
<td>0.00-0.24</td>
<td>Will not occur during program</td>
</tr>
</tbody>
</table>

Table 3  Risk Effects Criteria

<table>
<thead>
<tr>
<th>CAT TITLE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Catastrophic</td>
<td>Program in jeopardy of termination</td>
</tr>
<tr>
<td>Serious</td>
<td>Serious damage to program</td>
</tr>
<tr>
<td>Moderate</td>
<td>Problems cause program focus difficulties</td>
</tr>
<tr>
<td>Minor</td>
<td>Problems that can be easily be overcome</td>
</tr>
<tr>
<td>Null</td>
<td>No problem</td>
</tr>
</tbody>
</table>

Figure 37  Risk Matrix
If you were to compare this information with the MIL-STD-882 safety hazard matrix, you would find that the safety hazard matrix offered in the military standard uses letters for one of the two axes and that the highest hazards (risks) have the lowest indices. Our matrix in Figure 37 uses numbers on both axes and the index values are higher for more serious risks. Therefore, it is possible to apply the index values in a mathematical sense as a program metric. Given the 6 program risks listed in the program risk list shown in Table 4 with the indicated risk index values, the instantaneous program risk index is 97.

Table 4  Program Risk List

<table>
<thead>
<tr>
<th>RISK NBR</th>
<th>RISK TITLE</th>
<th>PROB</th>
<th>EFF</th>
<th>INDEX</th>
<th>TM</th>
<th>PRINCIPAL</th>
<th>SUSP</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Life Cycle Cost</td>
<td>4</td>
<td>4</td>
<td>16</td>
<td>1</td>
<td>Burns</td>
<td>02-10-05</td>
</tr>
<tr>
<td>5</td>
<td>Payload Capacity</td>
<td>4</td>
<td>5</td>
<td>20</td>
<td>1</td>
<td>Adams</td>
<td>03-08-05</td>
</tr>
<tr>
<td>7</td>
<td>Stoddard Supplier Risk</td>
<td>5</td>
<td>4</td>
<td>20</td>
<td>3</td>
<td>Thornton</td>
<td>04-20-05</td>
</tr>
<tr>
<td>12</td>
<td>Program Funding</td>
<td>4</td>
<td>4</td>
<td>16</td>
<td>0</td>
<td>Connolly</td>
<td>03-10-05</td>
</tr>
<tr>
<td>15</td>
<td>Computer Software Schedule</td>
<td>5</td>
<td>5</td>
<td>25</td>
<td>4</td>
<td>Sampson</td>
<td>05-23-05</td>
</tr>
<tr>
<td></td>
<td><strong>CURRENT PROGRAM INDEX</strong></td>
<td></td>
<td></td>
<td><strong>97</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Thus, we have arrived at a program risk index or metric. If we maintain a chart of this metric over time we see that it characteristically will rise early in a program as risks are identified but there is a delay in mitigating them causing a rising metric as shown in Figure 38. As a program progresses, this value will rise to a peak at some point in the program and subsequently start a long decline. Risks continue to be identified so we see the accumulated total number of risks continue to increase but the program is being successful in mitigating risks and later risks are commonly lower in index than those identified earlier in the program.

Figure 38  Program Risk Tracking Chart
3.5 Verification Requirements

The author respects the six-section specification structure of the DoD specification standard where Section 3 contains the product requirements and Section 4 the verification requirements. For every product requirement in Section 3 there should be one or more verification process requirements in Section 4. Whoever writes the Section 3 requirement should also write the verification requirement and with little time between the two actions. The rationale here is that the author of an unverifiable requirement will have some much difficulty writing a verification requirement and this difficulty may stimulate them to look for better ways of writing the requirement leading to a verifiable requirement.

We commonly respect four methods of verification: test, analysis, demonstration, and examination. A verification traceability matrix should be included in every specification that correlates every requirement in Section 3 with a method of verification and one or more verification requirements in Section 4. Each one of the rows in the verification traceability matrix forms a verification string. All of the verification traceability matrices are pooled into a single program-wide verification compliance matrix shown in Figure 39 that lists every verification string.

Figure 39 Verification Traceability
A verification engineer or team must now assign verification task numbers to all of the strings in the compliance matrix. Each task is identified in a task matrix and coordinated with the name of a principal engineer who must plan the task, make arrangements for the needed resources in a timely way relative to the task, accomplish the task on schedule, and produce a verification report. The reports from all of the tasks for a given item may be subjected to a configuration audit by the customer to ensure that the contractor did meet all of the requirements in the specification.

3.6 Specification Review and Approval

No matter the path the specification has taken through the requirements analysis process relative to modeling, the program should pass it through a review and approval process before it becomes a part of the formal configuration baseline definition. The review and approval process, shown in Figure 40, offers a formal or informal peer review way of comparing the content of a specification with a set of standards that all specifications should meet. Following approval, the specification must be formally released, published, and made available to program personnel either in paper or on-line form. The released specification must thereafter be formally protected through configuration management. Any changes must pass back through this same process to gain approval.

![Figure 40 Specification Review and Approval](image)

The formal review process should include a conscious evaluation of specification template faithfulness and overall quality measured in accordance with a specification checklist. Next, the specification should be checked for adherence with good traceability standards. The program may choose to fully implement traceability standards shown in the figure or some subset thereof. The final string of checks shown in Figure 40 assesses the specification for residual risk, completeness and excess content. A decision is reached by the reviewers followed by the review chairman calling either for corrective action or approval of the specification, if needed.

Specifications prepared on small or advanced programs may not have sufficient budget to support a formal review process. In this case, while not as supportive of a low risk approach, a
specification can be reviewed by experienced people in a less organized fashion, called a peer review. The team is assembled and asked to review the document either together or on-line at their desk followed by a group meeting to discuss the content.

The master copy of each specification must be retained and protected by an assigned authority in order to protect the integrity of the document. Once approved and released, this master must be accurately identified and protected against change. In one organization the author recalls, the master was changed during work on an engineering change proposal (ECP) but the ECP was subsequently canceled. The organization no longer had a master for the specification in effect because it had become corrupted by the change work that did not materialize. It helps to consider each specification build or change a separate campaign that results in the release of a document that will exist forever. If subsequently that document is changed, the change is built anew on the preserved baseline past.

Specifications must be readily available to personnel working a program. As they are released they must be distributed to those who need them. As they are revised the same is true of the revisions. Years ago specifications were crafted with typewriters and type setting. These were published in paper form and distributed using shoe leather and mail systems. If most of your specifications are in paper media, you may have no near term option but to place them in a paper document library from which program personnel may check them out physically. But, even if this is the current case, you should be making plans to move to an on-line specification library for cost, efficiency, and document configuration control purposes.

In a paper media, after a specification is formally released, the master must go to reproduction where sufficient copies are made to cover the needs for distribution and the library. The master should be returned to what the author has accurately heard referred to as the vault, a physically secure facility (not in the classified data sense unless this is also a valid concern) where all of the engineering masters are retained. The copies must then be physically distributed. If the specification in question is also a customer-approved specification, another loop will be required to gain their approval prior to distribution. A networked library will avoid a great deal of this busy work.

Adios paper and good riddance. Even if you are currently using a paper media for distribution you probably already have the resources in place to convert to networked computer media. It requires specifications captured in computer file media, a network with adequate storage capacity and speed, and easy access from work stations on the part of the people. These features are present in most everyone’s shop today or are not beyond the pale to achieve. It is unimaginable that anyone would use a typewriter today to prepare a specification so they will always be created in some form of word processor or computer database. The results of this work can be passed on to the document release function on a disk or via the network connection and thereafter transferred to an on-line library from which anyone may open it but not change it.
AN EFFECTIVE SPECIFICATION DEVELOPMENT ALGORITHM
STUDENT MANUAL

EXHIBIT A
PRESENTATION MATERIALS
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, AGM-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
Valuation and Verification, System Engineering Planning and
Enterprise Identity, System Engineering Deployment
An Effective Specification Development Algorithm Tutorial Outline

<table>
<thead>
<tr>
<th>2.1</th>
<th>Specification Preparation Readiness</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.2</td>
<td>Specification Preparation Readiness</td>
</tr>
<tr>
<td>2.3</td>
<td>Traditional Structured Analysis</td>
</tr>
<tr>
<td>2.4</td>
<td>Traditional Structured Analysis</td>
</tr>
<tr>
<td>2.5</td>
<td>Unified Modeling Language</td>
</tr>
<tr>
<td>2.6</td>
<td>Unified Modeling Language</td>
</tr>
<tr>
<td>2.7</td>
<td>Requirements Management</td>
</tr>
<tr>
<td>2.8</td>
<td>Requirements Management</td>
</tr>
</tbody>
</table>

Success Is Possible

- **The Goal**
  - Good specifications
  - On time
  - Affordable

- **The Plan**
  - A sound beginning - be prepared
  - A clear path to a successful state - always clear what must be done
  - An effective closing - a specification review and approval process
What is a Specification?

A specification contains all of the requirements for a given item.

The Word Requirement, From The Dictionary

- Something wanted or necessary.
- Something essential to the existence or occurrence of something else.
- A necessary characteristic or attribute of some thing (or item).
In Writing a Specification, What Is the Target?

How to Hit the Target of Minimized Completeness

- Every performance requirement traceable to a model from which it was derived
- Every external interface for the item identified and defined in interface requirements in the specification (unless ICD applied)
- Every specialty engineering discipline that has been mapped to the item is included in the specification
- Every environmental influence defined in the appropriate model (system, end item, component) mapped to appropriate specification content.
- Every requirement in the specification traceable to a parent item specification requirement (ideally applies to the system specification relative to user requirements as well).
- Requirements are quantified as appropriate to the statement.
- Requirements are validated (risks understood and mitigated).
Product Requirements Types

• Hardware
  – Performance
  – Constraints
    • Interface
    • Specialty Engineering
    • Environmental

• One view of software requirements
  – Functional
  – Non-Functional
  – Quality

• My view of software types
  – Same as systems and hardware

Requirements Types

All of these requirements must be identified before product and process detailed design work is started and they must be mutually consistent.
Attributes of a Good Requirement

- Achievable (validated)
- Quantified
- Achievable (validated)
- Verifiable/testable
- Unambiguous
- Complete (covers all cases)
- Performance specification
  - Design independent
  - What, not how
- Detail specification
  - Design dependent

Some Good Examples

Frequency coverage. Item frequency coverage shall be 225.0 to 399.9 Megahertz inclusive in tenth Megahertz steps.

Weight. Item weight shall be less than or equal to 240 pounds.

Range. Maximum achievable range shall be greater than or equal to 5,000 nautical miles while recognizing a fuel loading safety margin of 10% or more.

Range. Maximum range shall be greater than or equal to 2,500 nautical miles.

Reliability. Item MTBF shall be greater than or equal to 10,000 hours.
Some Bad Examples

- The screens used in the system shall be designed in a user friendly manner.
- Item weight shall not be greater than 153 pounds.
- Aircraft shall identify their position within 1000 feet of actual along and across track position using Loran C.
- Brakes shall function smoothly and stop the train in a safe distance.
- There shall be no hailstorms in the path of the aircraft.
- On most days, transmitter power output should be 100 watts.
- Go fast.
- Item shall work well and last a long time.
- Any favorites from your past?

Specifications Are Full of Sentences

- THESE SENTENCES SHOULD BE WRITTEN IN THE SIMPLEST POSSIBLE WAY
- THE SUBJECT IS THE ITEM CHARACTERISTIC ABOUT WHICH THE REQUIREMENT IS WRITTEN
- VERB SHALL CLEARLY CALLS FOR COMPLIANCE

<table>
<thead>
<tr>
<th>SUBJECT</th>
<th>VERB</th>
<th>OBJECT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Item memory margin</td>
<td>shall be</td>
<td>greater than 100%</td>
</tr>
</tbody>
</table>
The Verb

- SHALL  Mandatory
- WILL   Contractor intends to perform
- SHOULD Recommended, Desirable

The Subject

- Writing requirements is easy
- The difficult job is knowing what to write them about - the subject of the sentence
- That is why we model the problem space
Good Examples In Primitive Form

- weight $< 240$ pounds
- range $> 5,000$ nautical miles
- MTBF $> 10,000$ hours

Requirements Analysis Strategies

- Modeling
- Structured Decomposition
- Architecture Synthesis
- Item Identification
- Freestyle
- Cloning
- Component Standardization
- Customer Q&A (Elicitation)
- Freestyle is for Experts and Fools
A Foolproof Search For Subjects

Structured Modeling Tools

Specification Template

Models

System Definition Document

Primitive List

Item Specification

Language, Style, Format

Essential Characteristics

General Program Task-Resource Relationships

Program Management Skill

Program Process Task

Program Management Data

Program Process Audit

Program Process

People with Experience

Common Process Metrics

Feedback

Continuous Improvement Process

People with Experience

Program Process

Process

Product

Disposal

Work Product

Program Material

Product

Material

External Standards

Resources

Program Plans

Task Controls

Customer Needs

Functional Department Process

Functional Skill

Qualified People

Tools

Training

Typical DID-Template-Work Product Channel
Document Progressions

System Development Process Overview

Covered in this tutorial
Preparatory Steps

A Single Model Will Not Work
Hardware and Systems Analysis Models

- Traditional structured analysis
  - Functional analysis
    - Functional flow diagramming
    - Enhanced functional flow diagramming (CORE)
    - Behavioral diagramming (RDD/IPO)
    - IDEF 0 (SADT)
    - Process flow analysis
    - Hierarchical functional analysis
  - Constraints analysis
    - State diagramming
    - SysML

Computer Software Structured Analysis Models

- Process-oriented analysis
  - Flow charting
  - Modern Structured Analysis (Yourdon-Demarco)
  - Modern Structured Analysis (Hatley-Pirbhai)
- Data-oriented analysis
  - Table normalizing
  - IDEF-1X
- Object-oriented analysis
  - Early models
    - Unified Modeling Language (UML)
- DoD architecture framework (DoDAF)
Structured View of a Problem Space

Structured Analysis Methods Comparison

<table>
<thead>
<tr>
<th>MULTI-FACETED APPROACHES</th>
<th>PRODUCT ENTITY FACET</th>
<th>FUNCTIONAL FACET</th>
<th>BEHAVIORAL FACET</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRADITIONAL STRUCTURED ANALYSIS</td>
<td>PRODUCT ENTITY BLOCK DIAGRAM</td>
<td>FUNCTIONAL FLOW DIAGRAM</td>
<td>SCHEMATIC BLOCK DIAGRAM</td>
</tr>
<tr>
<td>MODERN STRUCTURED ANALYSIS</td>
<td>HIERARCHICAL DIAGRAM</td>
<td>DATA FLOW DIAGRAM</td>
<td>P SPEC, STATE DIAGRAM</td>
</tr>
<tr>
<td>EARLY OBJECT-ORIENTED ANALYSIS</td>
<td>CLASS AND OBJECT DIAGRAM</td>
<td>DATA FLOW DIAGRAM</td>
<td>STATE DIAGRAM</td>
</tr>
<tr>
<td>UML</td>
<td>CLASS/OBJECT, COMPONENT, &amp; DEPLOYMENT DIAGRAMS</td>
<td>USE CASES AND ACTIVITY DIAGRAMS</td>
<td>STATE, SEQUENCE, AND COMMUNICATION DIAGRAMS</td>
</tr>
</tbody>
</table>

° UNPRECEDENTED ANALYTICAL ENTRY FACET
## Model Suggestions for Today

<table>
<thead>
<tr>
<th>SPECIFICATION TYPE</th>
<th>MODEL SUGGESTED</th>
</tr>
</thead>
<tbody>
<tr>
<td>System Specification</td>
<td>Traditional Structured Analysis</td>
</tr>
<tr>
<td>Hardware Performance Specification</td>
<td>Traditional Structured Analysis</td>
</tr>
<tr>
<td>Software Performance Specification, General</td>
<td>Unified Modeling Language (UML)</td>
</tr>
<tr>
<td>Software Performance Specification, Database</td>
<td>DoDAF</td>
</tr>
<tr>
<td>Software Performance Specification, DoD IS</td>
<td></td>
</tr>
</tbody>
</table>

But, be prepared to move to the use of SysML coupled with UML and the eventual merge of the two into a more fully integrated common modeling method.

## Preparatory Steps
Enabling Documentation

EEQ defines the enterprise common process to be applied on all programs, the functional departments, and their process responsibilities which include preparing a department manual covering all common process work allocated to the department.

The system engineering department manual that defines the way system engineering will be applied on all programs including requirements analysis and specification management.

All functional departments should define templates and data item descriptions for all work products that are prepared as documents. One example of these artifacts is a set of specification data item descriptions that are coordinated with a particular template and a method of acquiring the content through modeling.

Preparatory Steps
MIL-STD-961E Template

Recommended Template With Map for Traditional Structured Analysis
## Recommended Template With Map for Traditional Structured Analysis

<table>
<thead>
<tr>
<th>PARA</th>
<th>TITLE</th>
<th>MODEL</th>
<th>DEPT</th>
<th>APP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>SCOPE</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>2.</td>
<td>APPLICABLE DOCUMENTS</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.</td>
<td>REQUIREMENTS</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1</td>
<td>Functional and performance requirements</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.1</td>
<td>Threat</td>
<td>Mission Analysis</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.3</td>
<td>Required states and modes</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.3.1</td>
<td>Functional analysis</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.3.2</td>
<td>Subordinate entities</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.3.3</td>
<td>Interface relationships</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.3.4</td>
<td>Specialty engineering requirements</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.1.3.5</td>
<td>Environmental model</td>
<td>DID</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.2</td>
<td>Entity capability requirements</td>
<td>Functional Analysis</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.2.m</td>
<td>Capability m</td>
<td>Functional Analysis</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.2.m.n</td>
<td>Capability m, requirement n</td>
<td>Functional Analysis</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>3.3</td>
<td>Interface requirements</td>
<td>N-Square Diagram</td>
<td>2100</td>
<td>D</td>
</tr>
<tr>
<td>3.3.1</td>
<td>External interface requirements</td>
<td>N-Square Diagram</td>
<td>2100</td>
<td>D</td>
</tr>
<tr>
<td>3.3.1.m</td>
<td>External interface m</td>
<td>N-Square Diagram</td>
<td>2100</td>
<td>D</td>
</tr>
<tr>
<td>3.3.2</td>
<td>Internal interface requirements</td>
<td>N-Square Diagram</td>
<td>2100</td>
<td>D</td>
</tr>
<tr>
<td>3.3.2.m</td>
<td>Internal interface m</td>
<td>N-Square Diagram</td>
<td>2100</td>
<td>D</td>
</tr>
<tr>
<td>3.3.2.m.n</td>
<td>Internal Interface m, requirement n</td>
<td>N-Square Diagram</td>
<td>2100</td>
<td>D</td>
</tr>
</tbody>
</table>

---

**Modeling Methods Map**

**Functional Department Map**

**Model Artifact Capture Map**

---

A2-1-19
Prepare and Maintain DIDs

- The DID follows the template format
- The recommended DID is focused on a particular modeling approach
- The DID tells how to create a specification using that template with a particular modeling approach

Here is a sample DID for a system specification using TSA.

Preparatory Steps
## Generic Specialty Engineering Scoping Matrix

### H - Reliability
- **Department:** D2164
- **Para:** 3.4.1

### H2 - Maintainability
- **Department:** D2164
- **Para:** 3.4.2

### H3 - Availability
- **Department:** D2164
- **Para:** 3.4.3

### H4 - Deployability and transportability
- **Department:** D2164
- **Para:** 3.4.4

### H5 - Logistics
- **Department:** D2311
- **Para:** 3.4.5

### H6 - Maintenance
- **Department:** D2311
- **Para:** 3.4.5.1

### H7 - Interchangeability
- **Department:** D2113
- **Para:** 3.4.5.2

### H8 - Supply
- **Department:** D2311
- **Para:** 3.4.5.3

### H9 - Facilities and facility equipment
- **Department:** D2311
- **Para:** 3.4.5.4

### HA - Personnel
- **Department:** D2313
- **Para:** 3.4.5.5

### HB - Training
- **Department:** D2313
- **Para:** 3.4.5.6

### HC - Safety
- **Department:** D2165
- **Para:** 3.4.6

### HD - Human factors engineering
- **Department:** D2160
- **Para:** 3.4.7

### HE - Security and privacy
- **Department:** D2160
- **Para:** 3.4.8

### HF - Electromagnetic compatibility
- **Department:** D2136
- **Para:** 3.4.9

### HG - Lightning protection
- **Department:** D2136
- **Para:** 3.4.10

### HH - Productivity
- **Department:** D2160
- **Para:** 3.4.11

### HI - Affordability
- **Department:** D2160
- **Para:** 3.4.12

### HJ - Computer resource requirements
- **Department:** D2160
- **Para:** 3.4.13

### HK - Quality Engineering
- **Department:** D5100
- **Para:** 3.4.14.1

### HL - Parts, materials, and processes
- **Department:** D2167
- **Para:** 3.4.14.2

---

## Generic Specialty Engineering Scoping Matrix

### HM - Workmanship
- **Department:** D5100
- **Para:** 3.4.14.3

### HK - Nameplates and product markings
- **Department:** D2113
- **Para:** 3.4.14.4

### HJ - Serializatation
- **Department:** D2113
- **Para:** 3.4.14.5

### HK - Mass properties
- **Department:** D2124
- **Para:** 3.4.14.6

### HQ - Structural properties
- **Department:** D2124
- **Para:** 3.4.14.7

### HR - Shock and vibration
- **Department:** D2124
- **Para:** 3.4.14.8

### HS - Earthquake survivability
- **Department:** D2123
- **Para:** 3.4.14.9

### HT - Aerodynamics
- **Department:** D2144
- **Para:** 3.4.14.10

### HU - Thermodynamics
- **Department:** D2143
- **Para:** 3.4.14.11

### HV - Chemical, electrical, and mechanical properties
- **Department:** D2167
- **Para:** 3.4.14.12

### HW - Stability
- **Department:** D2167
- **Para:** 3.4.14.13

### HX - Coatings and Corrosion Control
- **Department:** D2167
- **Para:** 3.4.14.14
Proposal Team Specification Actions

Environmental Requirements

- **System**
  - Identify spaces within which the system will have to function
  - Select standards covering those spaces
  - For each standard, select parameters that apply
  - Tailor the range of selected parameters

- **End item**
  - Build three dimensional model of end items, physical processes, and process environments
  - Extract item environments

- **Component**
  - Zone end item into spaces of common environmental characteristics
  - Map components to zones
  - Components inherit zone environmental requirements
Specialty Engineering Scoping Matrix
Applied to Program

Configuration Control the Models
On To Program Work
AN EFFECTIVE SPECIFICATION DEVELOPMENT ALGORITHM TUTORIAL

TRADITIONAL STRUCTURED ANALYSIS

The Big Bang Theory Of System Development
THE TRADITIONAL APPROACH

EVERYTHING FLOWS FROM ONE IDEA,
THE CUSTOMER NEED
IT IS THE ULTIMATE REQUIREMENT,
THE ULTIMATE FUNCTION

BA-BA-BA-BANG
Two Top-Level Views of a System

TRADITIONAL STRUCTURED ANALYSIS (TSA)

ENVIRONMENT I2 SYSTEM I1

BORROWED FROM MODERN STRUCTURED ANALYSIS (MSA)

TERMINATOR 1
TERMINATOR 2
TERMINATOR 3
TERMINATOR 4
TERMINATOR 5
TERMINATOR 6

The Beginning Of Functional Decomposition

SYSTEM NEED STATEMENT

MINDLESS ALLOCATION

FUNCTIONAL DECOMPOSITION

CONTINUING FUNCTION ALLOCATION

SYSTEM ARCHITECTURE DEFINITION
Traditional Structured Analysis Model

Overview

1. Understand User Requirements
2. Functional Flow Diagram
3. Decomposition
4. Performance Requirements Analysis
5. Requirement Allocation
6. Product Entity Structure
7. Environmental Requirements Analysis
8. N-Square Diagram
9. Requirement Allocation Sheet
10. Specifications Cycle to Lower Tiers
11. Speciality Engineering Requirements Analysis

The Ultimate Function and Its First Expansion

First Expansion is a Life-Cycle Flow Diagram

Alternative functional analysis techniques

- Enhanced functional flow block diagramming (CORE)
- Behavioral diagramming (RDD)
- IDEF-0
Use System Expansion Example
Space Transport System
Continued Function Decomposition

An orderly exposure of needed functionality moving from the known to the unknown, from simple to the complex, from the top to the bottom.

Performance Requirements Analysis
and the RAS

<table>
<thead>
<tr>
<th>FUNCTION</th>
<th>PRODUCT ENTITY</th>
<th>REQUIREMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>MID NAME</td>
<td>PID NAME</td>
<td>RD STATEMENT</td>
</tr>
<tr>
<td>F123</td>
<td>A12 Engine</td>
<td>WEBBFS</td>
</tr>
</tbody>
</table>

Exposing what the system must do and how well it must do it encouraging identification of all essential characteristics and avoidance of unnecessary characteristics.
The Function-Product Entity Plane

Functional Analysis Alternatives

- IDEF 0
  - A variation on SADT
- Behavioral Diagramming
  - From Ascent Logic’s RDD
  - Based on IPO
- Enhanced Functional Flow Block Diagramming
  - Employed in Vitech’s CORE
- Hierarchical Functional Block Diagramming
Requirements Capture Using the RAS-Complete Format

Here Is What We Want

This Is How To Get It

ENVIRONMENTAL REQUIREMENTS ANALYSIS

PRODUCT ENTITY IDENTIFICATION

FUNCTIONAL ANALYSIS

TIMELINE ANALYSIS

IN-SERVICE ANALYSIS

SPECIALTY ENGINEERING REQUIREMENTS ANALYSIS

ENVIRONMENTAL STANDARDS SELECTION AND TRAJECTORY

SERVICE USE PROFILE ANALYSIS

END-STATE CODING ANALYSIS

<table>
<thead>
<tr>
<th>MOD/NAME</th>
<th>REQUIREMENT ID</th>
<th>REQ. NAME</th>
<th>TEXT</th>
<th>MOD/NAME</th>
<th>PARA</th>
<th>TITLE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PERFORMANCE REQUIREMENTS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>INTERFACE REQUIREMENTS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SPECIALTY ENGINEERING REQUIREMENTS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SYSTEM ENVIRONMENTAL REQUIREMENTS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>END-STATE ENVIRONMENTAL REQUIREMENTS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>COMPONENT ENVIRONMENTAL REQUIREMENTS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Product Entity Structure

Use a common structure that includes hardware and software.
Organizing For Interface Development

- Decompose needed functionality and allocate to product entities
- Map product entities to responsible development organizations
  - Create cross-functional integrated product and process teams (IPPT)
  - Assign principal engineers for lowest tier responsibilities on teams (everything has someone responsible)
- Establish clear rules for interface development responsibility
  - Identify needed interfaces as a function of how functionality was allocated to entities
  - Analyze product entity pair relationships using n-square diagrams
  - Partition interface into subsets as a function of product entity principal views
  - Assign interface responsibility to product entity principal engineers as a function of a receiving terminal rule (if you need an interface you must come forward)
  - System engineering manage the aggregate external and inter-team interface sets applying a lowest common team integration concept
- Minimize external (cross-organizational) interface at all levels, iterating product entity structure and/or development organization responsibilities to do so, if necessary, then apply system engineering integration resources to that which remains
Two Interface Definition Models

SCHEMATIC BLOCK DIAGRAMMING

• Lines define interfaces
• Blocks are objects only from the product entity structure

N-SQUARE DIAGRAMMING

• Marked intersections define interfaces
• Diagonal blocks are objects only from product entity block diagram
• Apparent ambiguity reflects directionality

Interface Requirements Derivation
Geometrical View
Development Often Fails at the Cross-organizational Interfaces

Interface Integration Focus
The Fundamental Problems in Interface Work

There is a one-to-one correspondence between teams and components. There is a one-to-two correspondence between teams and interfaces.

We tend to focus inwardly
The Fundamental Problems in Interface Work

We are dependent on the worst interface on planet Earth in the development of interfaces.

Benefits Of Product Team Organization
Specialty Engineering Identification of Constraints

SPECIALTY ENGINEERING CONSTRAINTS MATRIX

SPECIALTY ENGINEERING REQUIREMENTS FLOW INTO THE INDICATED SPECIFICATIONS VIA THE RAS IMPLEMENTED IN A DATABASE

Specialty Engineering Plane Added
Environment Subsets

Environmental Requirements

- **System**
  - Identify spaces within which the system will have to function
  - Select standards covering those spaces
  - For each standard, select parameters that apply
  - Tailor the range of selected parameters

- **End item**
  - Build three dimensional model of end items, physical processes, and process environments
  - Extract item environmental requirements

- **Component**
  - Zone end item into spaces of common environmental characteristics
  - Map components to zones
  - Components inherit zone environmental requirements
Environmental Planes Added

RAS Complete In Tabular Form

<table>
<thead>
<tr>
<th>Performance Requirements</th>
<th>Requirement Entry Name</th>
<th>Requirement Description</th>
<th>Product Entity Name</th>
<th>Requirement Entity Name</th>
<th>Requirement Entity Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specialty Engineering</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Interface</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Environmental</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A-2-32

2R Short

AOG System Engineering, Inc.
AN EFFECTIVE SPECIFICATION DEVELOPMENT ALGORITHM TUTORIAL

UNIFIED MODELING LANGUAGE

Agenda

• The Dead Sea scrolls of software development
• UML modeling artifacts
• UML modeling approach
• Integration
• Requirements and modeling documentation
Software Dead Sea Scrolls

- Flow chartings
  - Functionality examined
  - Data in the back seat
- SADT and IPO
  - Two axis models
- Modern structured analysis and HP
  - Functionality and data examined
- Early OOA
  - Search for objects and their behavior
  - Anti Sullivan

A Preferred Modeling Order

Early object oriented analysis encouraged this pattern.

We will follow Sullivan’s encouragement in this tutorial - form follows function.

UML can support either direction.

Note: A classifier is a general term for a software product entity represented by a node, component, or class in UML.
The Software Development Process

- Identify a product entity that will be developed as computer software.
- Dynamically analyze the entity.
  - Use cases
  - Sequence diagram
  - Communication diagram
  - Activity diagram
  - State diagram
- In the sequence, communication, and activity diagramming analysis you will have to identify lower tier 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.

Consolidated System Product Entity Structure
Suggested SRS Structure

3 REQUIREMENTS
3.1 Required states and modes
3.2 Software entity capability requirements
3.2.m Software entity capability m
3.2.m.n Software entity capability m, requirement n
3.3 Software entity interface requirements
3.3.1 Software entity external interface requirements
3.3.1.m Specific external interface m
3.3.1.m.n External interface m, requirement n
3.3.2 Software entity internal interface requirements
3.3.2.m Specific internal interface m
3.3.2.m.n Internal interface m, requirement n
3.3.3 Software entity internal data requirements
3.3.3.n Specific software entity internal data requirement n
3.4 Specialty engineering requirements
3.5 Software entity environmental requirements
3.6 Precedence and criticality requirements

The Diagrams of UML

• For modeling dynamic aspects of the system
  – Use case diagram
  – Sequence diagram
  – Timing diagram
  – Communication diagram (renamed in 2)
  – State diagram
  – Activity 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)

(2) = added in UML 2.0
**The Dynamic Models**

**Sequence Diagram UX-11321**
Emphasizes the time ordering of messages

- **Actor**
  - messageOne()
  - messageFive()

- **Classifier AX1**
  - messageTwo()
  - messageFour()

- **Classifier AX2**
  - messageThree()

**Time**

**Lifeline active**

**Argument List**

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

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

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
A Simple Example

Communication Diagram UX11322
Emphasizes structural relationships

Semantically identical to the sequence diagram.
The Static Entities in UML

- **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 node or collection of collection of nodes.

- **Node**
  - Appears on a deployment diagram that exists at run time and 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 classes.

- **Class**
  - A description of a set of objects that share the same attributes, operations, relationships, and semantics.

- **Object**
  - An instance of a class.

UML Structural Artifacts in a Product Entity Structure
Classes and Objects

A class is a description of a set of objects that share the same attributes, operations, relationships, and semantics. An object is an instance of a class. Graphically a class is rendered as a rectangle.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>The name is a noun or noun phrase</td>
</tr>
<tr>
<td>attributeOne</td>
<td>An attribute is a named property of a class that describes a range of values that instances of the property may hold. An attribute represents some property of the thing you are modeling that is shared by all objects of that class.</td>
</tr>
<tr>
<td>attributeTwo</td>
<td></td>
</tr>
<tr>
<td>attributeThree</td>
<td>An operation is the implementation of a service that can be requested from any object of a class to effect behavior. An operation is an abstraction of something you can do to an object that is shared by all objects of that class.</td>
</tr>
<tr>
<td>attributeFour</td>
<td></td>
</tr>
<tr>
<td>operationOne()</td>
<td></td>
</tr>
<tr>
<td>operationTwo()</td>
<td></td>
</tr>
<tr>
<td>operationThree()</td>
<td></td>
</tr>
</tbody>
</table>
Class Responsibilities

A responsibility is a contract or an obligation of a class. You may find it useful to begin the analysis of classes this way translating these into attributes and operations that best fulfill the class’s responsibility as the model is refined. The responsibility is noted in an added compartment in which descriptive free form-text is entered.

Structural Relationships

These Have Nothing To Do With Messages

NameOne is the base class. NameTwo and NameThree are leaf classes in a generalization.

NameOne depending on NameFour for information and services.

An association is a structural relationship
Structural Relationships
Association Adornments

- **Association Name**
  - NameOne
  - NameTwo
  - Association name
  - name direction

- **Association Role**
  - The face that the class at the far end of an association presents to the class at near end of the association. Role names called end names.

- **Association Multiplicity**
  - Tells how many objects may be connected across an association instance. Given by a range of numbers.

- **Association Aggregation**
  - Expresses a whole-part relationship between to associated classes.

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 use case.
- For each scenario build a sequence diagram and in the process identify next lower tier classifiers and messages between the actors and lower tier classifiers.
- Apply communication, activity, and state diagrams as needed.
- Derive requirements from dynamic modeling artifacts.

Hierarchical Modeling Relationships

A top-level software product entity

CLASSIFIER AX

CONTEXT DIAGRAM TERMINATOR

USE CASE

EXTENDED USE CASE

SCENARIO

SEQUENCE DIAGRAM UX-hijk1

COMMUNICATION DIAGRAM UX-hijk2

ACTIVITY DIAGRAM UX-hijk3

STATE DIAGRAM UX-hijk4
The classifier is the product entity the specification is being written for.
The terminators reflect necessary external influences between the system and its environment.

Use Case Fundamentals

- A use case is a more expressive context diagram common 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

• **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 UX-11

Actors derive benefits from the system.

The word extend is used here in a generic way 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
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.

Examine Each Scenario Dynamically

Activity, sequence, and communication diagrams require identification of lower tier entities leading to additional of entities on the consolidated product entity diagram.

State diagrams may also be useful in identifying essential characteristics appropriate for the entity being analyzed.

Requirements flow out of the dynamic analysis and into the specification for the entity being analyzed.
AN EFFECTIVE SPECIFICATION DEVELOPMENT ALGORITHM TUTORIAL

REQUIREMENTS MANAGEMENT

Agenda

• Process summary
• Organizing and specification responsibility
• Requirements risk management
• Traceability
• Tools
• Review, approval, release, and distribution
• A peek into the future
Life Cycle Model

Suggested Enterprise Structure
Benefits of Product-Oriented Team Structure

The System Product Entity Structure and Teaming
Risk Defined

- The danger that injury, damage, or loss will occur
- Somebody or something likely to cause injury, damage, or loss
- The probability, amount, or type of possible loss incurred by an insurer
- The possibility of loss in an investment or speculation
- The statistical chance of danger from something, especially from the failure of an engineered system

Risk Measurement Parameters

### Probability of Occurrence

<table>
<thead>
<tr>
<th>CAT</th>
<th>TITLE</th>
<th>P(O)</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Nearly Certain</td>
<td>0.95-1.00</td>
<td>Will occur at least once during program</td>
</tr>
<tr>
<td>4</td>
<td>Probable</td>
<td>0.75-0.95</td>
<td>Will probably occur once during program</td>
</tr>
<tr>
<td>3</td>
<td>Possible</td>
<td>0.50-0.75</td>
<td>May occur during program</td>
</tr>
<tr>
<td>2</td>
<td>Unlikely</td>
<td>0.25-0.49</td>
<td>Will probably not occur during program</td>
</tr>
<tr>
<td>1</td>
<td>Nearly Impossible</td>
<td>0.00-0.24</td>
<td>Will not occur during program</td>
</tr>
</tbody>
</table>

### Seriousness of Effect

<table>
<thead>
<tr>
<th>CAT</th>
<th>TITLE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Catastrophic</td>
<td>Program in jeopardy of termination</td>
</tr>
<tr>
<td>4</td>
<td>Serious</td>
<td>Serious damage to program</td>
</tr>
<tr>
<td>3</td>
<td>Moderate</td>
<td>Problems cause program focus difficulties</td>
</tr>
<tr>
<td>2</td>
<td>Minor</td>
<td>Problems that can be easily be overcome</td>
</tr>
<tr>
<td>1</td>
<td>Null</td>
<td>No problem</td>
</tr>
</tbody>
</table>
### Risk Metric Values

<table>
<thead>
<tr>
<th>SERIOUSNESS OF THE EFFECT</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>High Risk</td>
<td>25</td>
<td>20</td>
<td>15</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>Medium Risk</td>
<td>20</td>
<td>16</td>
<td>12</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>Low Risk</td>
<td>15</td>
<td>12</td>
<td>9</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>8</td>
<td>6</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

Legend:
- High Risk
- Medium Risk
- Low Risk

### Aggregate Program Risk Index

![Graph showing Aggregate Program Risk Index]

- Peak Risk Point
- Program Risk Index
- Number of Risks
- Time

Legend:
- PEAK RISK POINT
- PROGRAM RISK INDEX
- NUMBER OF RISKS
- TIME
Traceability Forms

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

Traceability Integration

- **Lateral traceability** within TSA and UML independently should be no problem with all requirements derived from models
- **Vertical interface traceability** is decomposition driven within TSA and UML as well as across the HW-SW gap
- Environmental requirements are significantly different between HW and SW so traceability is not a significant issue between them
- Performance requirements across the HW-SW gap offer a vertical traceability challenge
The System Product Entity Structure

Level at which a subordinate software entity is identified

This is the kind of relationship of interest

Hardware entity
Software entity
System

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

<table>
<thead>
<tr>
<th>Requirements Derived From UML Modeling</th>
<th>Requirements Derived From TSA Modeling</th>
<th>ACTUAL RID EXAMPLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>R%1</td>
<td>R%1</td>
<td>RU7Z7H</td>
</tr>
<tr>
<td>R%2</td>
<td>R%2</td>
<td>R98ER6</td>
</tr>
<tr>
<td>R%3</td>
<td>R%3</td>
<td>R937YF</td>
</tr>
<tr>
<td>R%4</td>
<td>R%4</td>
<td>RJ866G</td>
</tr>
<tr>
<td>R%5</td>
<td>R%5</td>
<td>RJT66T</td>
</tr>
<tr>
<td>R%6</td>
<td>R%6</td>
<td>RGHT6T</td>
</tr>
<tr>
<td>R%7</td>
<td>R%7</td>
<td>RID87W</td>
</tr>
<tr>
<td>R%8</td>
<td>R%8</td>
<td>RBJS87</td>
</tr>
<tr>
<td>R%9</td>
<td>R%9</td>
<td>RL34DF</td>
</tr>
<tr>
<td>R%10</td>
<td>R%10</td>
<td>R456HD</td>
</tr>
</tbody>
</table>

Alternatively, one could rely upon experienced inspection without the organizing influence of the matrix.
Today’s Tools

Tools Integration
Tomorrow’s Tools

• Front end modeling tools
  – Use case modeling
  – Function/activity modeling
  – State modeling (behavioral modeling)
  – Sequence/timeline modeling
  – Product entity and interface modeling
  – Specialty engineering database linkage
  – Environmental coverage

• Connection of modeling to management database

• Big dumb database
  – Requirements capture
  – Traceability
  – Value management
  – Specification publishing

Tools Integration
Configuration Management of Requirements Documentation

- Requirements Analysis
- Database Content
- Publish and Release
- Library

Portion of database corresponding to released specifications and library content under formal configuration control.
Utility Of Computer Projection

- ON-LINE NETWORK CAPABILITY
- PUT THE PROJECTION CAPABILITY IN THE WORK AREA
- APPLY REAL-TIME CONCURRENT DEVELOPMENT (IPD)
- FORM AND REFORM BETWEEN MEETING AND INDIVIDUAL WORK QUICKLY

Specification Review and Approval Process
Evaluate for Template Faithfulness

- Compare specification cover data with template (standard)
- Compare specification paragraphing structure with template
- Compare specification style with template style guide

Individual Requirement Quality

- Spot check specification requirements for requirements quality checklist compliance
- Spot check specification for requirements quantification where appropriate
Section 2 Traceability

- All documents listed in Section 2 called somewhere in the specification
- All documents tailored, if necessary, to limit coverage to the application
- All documents called in the requirements listed in Section 2
- Spot check for excessively tailored standards which could be quoted instead of being called
- Ensure documents called are current and accepted authorities for the application

Completeness and Avoidance of Unnecessary Content

- Ask principal engineer how content was derived
  - If ad hoc, there should be concern
  - If through structured analysis, spot check how a few requirements were derived (ask to see the supporting modeling data)
- Ensure all requirements traceable to parent requirements
-
Residual Risk Evaluation

- All TBD/TBR are closed or, if not, are being carried as program or team risks
- An approved concept exists

Movement To Universal Method
UML and Functional Analysis

Unified Modeling Language (UML)

<table>
<thead>
<tr>
<th>STATIC DIAGRAMS</th>
<th>DYNAMIC DIAGRAMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEPLOYMENT DIAGRAM</td>
<td>COMPONENT DIAGRAM</td>
</tr>
<tr>
<td>OBJECT &amp; CLASS DIAGRAMS</td>
<td>STATE CHART</td>
</tr>
<tr>
<td>INTERACTION DIAGRAMS</td>
<td>COMMUNICATION DIAGRAM</td>
</tr>
<tr>
<td>SEQUENCE DIAGRAM</td>
<td>USE CASE DIAGRAM</td>
</tr>
<tr>
<td>ACTIVITY DIAGRAM</td>
<td></td>
</tr>
</tbody>
</table>

Traditional Structured Analysis
A Subset of UML?

Modeling Changes In the Near Term
System Modeling Evolution Timeline

RISE IN THE USE OF STRUCTURED ANALYSIS

NOW

MODEL DRIVEN DEVELOPMENT

DATABASE DRIVEN DEVELOPMENT

DOCUMENT DRIVEN DEVELOPMENT

1920 1970 1990 2010 2030

05-15-2002 DATA UNSUBSTANTIATED
DATES ARE APPROXIMATE

NDIA

Over the Hill and Through the Woods to Utopia

Flow Charting

Modern Structured Analysis

Early OOA

HP

UML

DoDAF

SysML

Utopia

Traditional Structured Analysis

FFBD

IDEF0

EFFBD

BD

HFA

1950s

2010s
Review and Summary

The target is completeness and avoidance of unnecessary content

Use models to identify essential characteristics

Do the analysis

Write requirements in primitive form based on essential characteristics identified through modeling.

Translate into full sentences and insert into paragraph structure.
Practical Six Sigma Tools for Systems Engineering

9th Annual Systems Engineering Conference
23-26 October 2006

Rick Hefner, Ph.D.
Director, Process Management
Northrop Grumman Corporation
Background

- Six Sigma has proven to be a powerful enabler for process improvement
  - CMMI adoption
  - Process improvement for measurable ROI
  - Statistical analysis

- This presentation will focus on practical tools and techniques for use by systems engineers

Agenda

- What is Six Sigma?
- How does it apply to systems engineering?
- Strategies and lessons learned
## Projects Have Historically Suffered from Mistakes

### People-Related Mistakes
1. Undermined motivation
2. Weak personnel
3. Uncontrolled problem employees
4. Heroics
5. Adding people to a late project
6. Noisy, crowded offices
7. Friction between developers and customers
8. Unrealistic expectations
9. Lack of effective project sponsorship
10. Lack of stakeholder buy-in
11. Lack of user input
12. Politics placed over substance
13. Wishful thinking

### Process-Related Mistakes
14. Overly optimistic schedules
15. Insufficient Risk Management
16. Contractor failure Insufficient planning
17. Abandonment of planning under pressure
18. Wasted time during the fuzzy front end
19. Shortchanged upstream activities
20. Inadequate design
21. Shortchanged quality assurance
22. Insufficient management controls
23. Premature or too frequent convergence
24. Omitting necessary tasks from estimates
25. Planning to catch up later
26. Code-like-hell programming

### Product-Related Mistakes
27. Requirements gold-plating
28. Feature creep
29. Developer gold-plating
30. Push me, pull me negotiation
31. Research-oriented development

### Technology-Related Mistakes
32. Silver-bullet syndrome
33. Overestimated savings from new tools or methods
34. Switching tools in the middle of a project
35. Lack of automated source-code control

### Standish Group, 2003 survey of 13,000 projects
- 34% successes
- 15% failures
- 51% overruns

Reference: Steve McConnell, Rapid Development
Many Approaches to Solving the Problems

- Which weaknesses are causing my problems?
- Which strengths may mitigate my problems?
- Which improvement investments offer the best return?
Approaches to Process Improvement

Data-Driven (e.g., Six Sigma, Lean)

- Clarify what your customer wants (Voice of Customer)
  - Critical to Quality (CTQs)
- Determine what your processes can do (Voice of Process)
  - Statistical Process Control
- Identify and prioritize improvement opportunities
  - Causal analysis of data
- Determine where your customers/competitors are going (Voice of Business)
  - Design for Six Sigma

Model-Driven (e.g., CMM, CMMI)

- Determine the industry best practice
  - Benchmarking, models
- Compare your current practices to the model
  - Appraisal, education
- Identify and prioritize improvement opportunities
  - Implementation
  - Institutionalization
- Look for ways to optimize the processes
What is Six Sigma?

- Six Sigma is a management philosophy based on meeting business objectives by reducing variation
  - A disciplined, data-driven methodology for decision making and process improvement
- To increase process performance, you have to decrease variation

```
Defects
To early Too late
Delivery Time
Spread of variation too wide compared to specifications
```

```
Defects
Too early Too late
Reduce variation
Delivery Time
Spread of variation narrow compared to specifications
```

- Greater predictability in the process
- Less waste and rework, which lowers costs
- Products and services that perform better and last longer
- Happier customers
A Typical Six Sigma Project in Systems Engineering

- The organization notes that systems integration has been problematic on past projects (budget/schedule overruns)
- A Six Sigma team is formed to scope the problem, collect data from past projects, and determine the root cause(s)
- The team’s analysis of the historical data indicates that poorly understood interface requirements account for 90% of the overruns
- Procedures and criteria for a peer review of the interface requirements are written, using best practices from past projects
- A pilot project uses the new peer review procedures and criteria, and collects data to verify that they solve the problem
- The organization’s standard SE process and training is modified to incorporate the procedures and criteria, to prevent similar problems on future projects
Roles & Responsibilities - Organizational Implementation

- **Champions** – Facilitate the leadership, implementation, and deployment
- **Sponsors** – Provide resources
- **Process Owners** – Responsible for the processes being improved
- **Master Black Belts** – Serve as mentors for Black Belts
- **Black Belts** – Lead Six Sigma projects
  - Requires 4 weeks of training
- **Green Belts** – Serve on improvement teams under a Black Belt
  - Requires 2 weeks of training
Applicability to Engineering

- **System engineering processes are fuzzy**
  - Systems engineering "parts" are produced using processes lacking predictable mechanizations assumed for manufacturing of physical parts
  - Simple variation in human cognitive processes can prevent rigorous application of the Six Sigma methodology
  - Process variation can never be eliminated or may not even reduced below a moderate level

- **Results often cannot be measured in clear $ savings returned to organization**
  - Value is seen in reduced risk, increased customer satisfaction, more competitive bids, …
How Six Sigma Helps Process Improvement

- PI efforts often generate have little direct impact on the business goals
  - Confuses ends with means; results measured in activities implemented, not results

- Six Sigma delivers results that matter to managers (fewer defects, higher efficiency, cost savings, …)

- Six Sigma concentrates on problem solving in small groups, focused on a narrow issue
  - Allows for frequent successes (3-9 months)

- Six Sigma focuses on the customer’s perception of quality
How Six Sigma Helps CMMI-Based Improvement

- For an individual process:
  - CMM/CMMI identifies what activities are expected in the process
  - Six Sigma identifies how they can be improved (efficient, effective)

Example – Project Planning in the CMMI
- Could fully meet the CMMI goals and practices, but still write poor plans
- Six Sigma can be used to improve the planning process and write better plans
How CMMI Helps Six Sigma Based Improvement

- CMM/CMMI focuses on organizational change
  - Provides guidance on many dimensions of the infrastructure

**Process Areas**
- Organizational Process Focus
- Organizational Process Definition
- Organizational Training
- Organizational Process Performance
- Organizational Innovation and Deployment

**Generic Practices (all process areas)**
- GP 2.1 Establish an Organizational Policy
- GP 2.2 Plan the Process
- GP 2.3 Provide Resources
- GP 2.4 Assign Responsibility
- GP 2.5 Train People
- GP 3.1 Establish a Defined Process
- GP 2.6 Manage Configurations
- GP 2.7 Identify and Involve Relevant Stakeholders
- GP 2.8 Monitor and Control the Process
- GP 3.2 Collect Improvement Information
- GP 2.9 Objectively Evaluate Adherence
- GP 2.10 Review Status with Higher-Level Management
Barriers and Challenges

- Capturing the first, “low hanging fruit” makes Six Sigma implementation look easy…
  - Clearer problems, simpler solutions, bigger payoffs
  - Little need for coordination

...but later projects are tougher

- Keeping projects appraised of similar efforts, past and current
- Focusing on “the pain”, not the assumed solution

- Engineering process measurements are often difficult to analyze
  - Dirty (or no) data, human recording problems
  - May necessitate Define-Measure-Analyze-Measure-Analyze-etc.

- Must demonstrate the value of quantitative data to managers
  - Management style - reactive vs. proactive vs. quantitative
  - Less value in a chaotic environment
  - Must engage customers
Additional Challenges

- Difficulty in collecting subjective, reliable data
  - Humans are prone to errors and can bias data
  - E.g., the time spent in privately reviewing a document

- Dynamic nature of an on-going project
  - Changes in schedule, budget, personnel, etc. corrupt data

- Analysis requires that complex SE processes be broken down into small, repeatable tasks
  - E.g., peer review

- Repeatable process data requires the project/organization to define (and follow) a detailed process
Tools & Techniques
DMAIC – A Structured Approach to Improving a Process

1. DEFINE
2. MEASURE
3. ANALYZE
4. IMPROVE
5. CONTROL
DMAIC – Define

- Purpose is to set project goals and boundaries
- Establishes upfront focus on customer
- Key products
  - Project charter
  - Process map
  - List of what is important to customer -- Critical to Quality factors (CTQs)
Identify Key Stakeholders Early On

- Develop communication plan based on level of commitment required

<table>
<thead>
<tr>
<th>Level of Commitment</th>
<th>Testers</th>
<th>Developers</th>
<th>Requirements Leads</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enthusiastic Support</td>
<td>O</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>Help it work</td>
<td>O</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Compliant</td>
<td>O</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hesitant</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Indifferent</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Uncooperative</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Opposed</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hostile</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Stakeholders (Examples)

- X = Current Level of Commitment
- O = Level of Commitment Necessary for Success
High level “as-is” process map
- 5 to 7 key steps of main action
- Used to focus on the fundamental elements of the process

**Suppliers-Inputs-Process-Outputs-Customers (SIPOC)**

- **Inputs**
  - Place Doc into Position in or on Copier
  - Set Number of Copies Needed
  - Set Size Required
  - Set Light/Dark Settings
  - Select Paper Tray/Source
  - Press “Copy” Button

- **Outputs**
  - Retrieve Copies
  - Output Tray

- **Customers**

- **Suppliers**
  - Product (y’s)
  - Correctly
  - Incorrectly
  - Size too Small
  - Size too Large

- **Process Parameters**
  - Noise Parameters
  - Controllable Process Parameters
  - SOP Parameters
  - Critical Parameters

- **KEY for (x’s)**
  - Product (y’s)
  - Copies with:
    - Machine producing copies
    - Machine not producing copies

- **Noise Parameters**
- **Controllable Process Parameters**
- **SOP Parameters**
- **Critical Parameters**

- **Suppliers-Inputs-Process-Outputs-Customers (SIPOC)**
Voice of the Customer

- CT "critical to" matrix links process or CT tree (columns of the matrix) and product or CTY tree (rows)
  - Critical To Satisfaction (CTS)
  - Critical To Quality (CTQ)
  - Critical To Delivery (CTD)
  - Critical To Cost (CTC)
  - Critical To Process (CTP) - Process parameters which significantly influence a CTQ, CTD, and/or CTC
DMAIC - Measure

- **Purpose is to narrow range of potential causes and establish a baseline capability level**
  - Identify specific problem(s)
  - Prioritize critical input/process/output measures
  - Validate measurement system

- **Key products**
  - Cause/effect diagrams
  - FMEA
  - Gage R&R
  - Data collection plan
  - Analysis results
### Failure Modes and Effects Analysis

- Used to identify the way in which errors happen; an error mode, the antithesis of function.
- Employed as a diagnostic tool in *improvement*.
- Used as a prevention tool in *design*.
- Deals with the three dimensions of an error mode:
  - Severity
  - Detectability
  - Frequency

---

<table>
<thead>
<tr>
<th>Process/Product Failure Modes and Effects Analysis (FMEA)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process or Product Name</td>
</tr>
<tr>
<td>Responsible:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Process Step/Part Number</th>
<th>Potential Failure Mode</th>
<th>Potential Failure Effects</th>
<th>Potential Causes</th>
<th>Current Controls</th>
<th>Actions Recommended</th>
<th>Resp</th>
<th>Actions Taken</th>
<th>S</th>
<th>E</th>
<th>V</th>
<th>D</th>
<th>C</th>
<th>T</th>
<th>D</th>
<th>E</th>
<th>P</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>---</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>---</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>---</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>---</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
</tbody>
</table>

---

NORTHROP GRUMMAN

Copyright 2005 Northrop Grumman Corporation
Data Collection Plan

Measurement Consistency and Accuracy

Data Collection Plan

<table>
<thead>
<tr>
<th>What questions do you want to answer?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

How will you ensure consistency and stability?

What is your plan for starting data collection?

How will the data be displayed?

- Meaning of measurement in relation to project, process, or product
- What is counted into or excluded
- Measurement validation method
  - Regular validation during collection
  - Periodic validation of samples or aggregates independent of collection tools and methods
- Frequency of measurement collection
- Calculations used to derive an indirect, aggregated, or accumulated value
- How and where measurements are stored and accessed
- Tools, methods, resources, and assignments required
DMAIC - Analyze

- Purpose is to evaluate data/information for trends, patterns, causal relationships and "root causes"

- Key products
  - Quantitative analysis results
  - Theory that has been tested
Six Sigma Tool Kit

Stratification

Queue 1
Queue 2

Hypothesis Testing

Chi-Square $\chi^2$
t-test ANOVA
Regression $Y = X_1$

Capability Analysis

Process Sigma = 2.7

Regression Analysis

Boxplot ANOVAs

Control Charts

Data Analysis

Copyright 2005 Northrop Grumman Corporation
Exercise – What is Quantitative Management?

- Suppose your project conducted several peer reviews of similar code, and analyzed the results
  - Mean = 7.8 defects/KSLOC
  - $+3\sigma = 11.60$ defects/KSLOC
  - $-3\sigma = 4.001$ defects/KSLOC

- What would you expect the next peer review to produce in terms of defects/KSLOC?
- What would you think if a review resulted in 10 defects/KSLOC?
- 3 defects/KSLOC?
Exercise - What is Required for Quantitative Management?

- What is needed to develop the statistical characterization of a process?
  - The process has to be stable (predictable)
    - Process must be consistently performed
    - Complex processes may need to be stratified (separated into simpler processes)
  - There has to be enough data points to statistically characterize the process
    - Processes must occur frequently within a similar context (project or organization)

![Individual Value Chart]

- What is needed to develop the statistical characterization of a process?

<table>
<thead>
<tr>
<th>Observation Number</th>
<th>Individual Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>3</td>
<td>9</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>6</td>
<td>9</td>
</tr>
<tr>
<td>7</td>
<td>12</td>
</tr>
<tr>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>9</td>
<td>12</td>
</tr>
<tr>
<td>10</td>
<td>7</td>
</tr>
<tr>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td>12</td>
<td>4</td>
</tr>
</tbody>
</table>

Mean = 7.8
UCL = 11.60
LCL = 4.001
What Is a Control Chart?

- A time-ordered plot of process data points with a centerline based on the average and control limits that bound the expected range of variation.

- Control charts are one of the most useful quantitative tools for understanding variation.
What Are the Key Features of a Control Chart?

![Chart showing key features of a control chart](chart.png)

- **Observation Number**
- **Mean = 7.8**
- **Upper Control Limit (UCL) = 11.60**
- **Lower Control Limit (LCL) = 4.001**
- **Process "Average"**
- **Time ordered x-axis**
- **Individual data points**
There are Many Types of Control Charts

Tests performed with unequal sample sizes
What is *Special Cause* and *Common Cause* Variation?

- **Common Cause Variation**
  - *Routine* variation that comes from within the process
  - Caused by the natural variation in the process
  - Predictable (stable) within a range

- **Special Cause Variation**
  - *Assignable* variation that comes from outside the process
  - Caused by an unexpected variation in the process
  - Unpredictable

---

**Observation Number**

**Individual Value**

<table>
<thead>
<tr>
<th>Observation Number</th>
<th>Individual Value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mean=7.8</td>
</tr>
<tr>
<td></td>
<td>UCL=11.60</td>
</tr>
<tr>
<td></td>
<td>LCL=4.001</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Observation Number</th>
<th>Individual Value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mean=7.467</td>
</tr>
<tr>
<td></td>
<td>UCL=13.36</td>
</tr>
<tr>
<td></td>
<td>LCL=1.578</td>
</tr>
</tbody>
</table>
What Is a **Stable** (Predictable) Process?

### U Chart of Defects Detected in Requirements Definition

- **U** = 0.07825
- **UCL** = 0.09633
- **LCL** = 0.06017

All data points within the control limits. No signals of special cause variation.
What if the Process Isn’t Stable?

- You may be able to explain out of limit points by observing that they are due to an variation in the process
  - E.g., peer review held on Friday afternoon
  - You can eliminate the points from the data, if they are not part of the process you are trying to predict

- You may be able to stratify the data by an attribute of the process or attribute of the corresponding work product
  - E.g., different styles of peer reviews, peer reviews of different types of work products
Hearing Voices

- **Voice of the process**
  = the natural bounds of process performance

- **Voice of the customer**
  = the goals established for the product/process performance

- **Voice of the business**
  = process performance needed to be competitive

- **Process capability may be determined for the**
  - Organization
  - Product line
  - Project
  - Individual

- **Typically, the higher the level of analysis, the greater the variation**
Common Challenges for Engineering

- Data are often discrete rather than continuous, e.g., defects
- Observations often are scarce
- Processes are aperiodic
- Size of the the object often varies, e.g., software module
- Data distributions may not be normal
How Do I Address These Challenges?

- Employ control chart types that specifically deal with discrete data distributions, e.g., \( u \)-charts and \( p \)-charts
- Use control charts that compensate for widely variable areas of opportunity
- Transform non-normal continuous data to normal data before constructing a control chart
- Cross check control charts with hypothesis tests where few data points exist
Typical Choices in Industry

- Most customers care about:
  - Delivered defects
  - Cost and schedule

- So organizations try to predict:
  - Defects found throughout the lifecycle
  - Effectiveness of peer reviews, testing
  - Cost achieved/actual (Cost Performance Index – CPI)
  - Schedule achieved/actual (Schedule Performance Index – SPI)

**Process performance**
- **Process measures** (e.g., effectiveness, efficiency, speed)
- **Product measures** (e.g., quality, defect density).
How Can Quantitative Management Help?

- By measuring both the mean and variation, the project/organization can assess the full impact of an “improvement”
  - Can focus on reducing the variation (making the process more predictable)
    - Train people on the process
    - Create procedures/checklists
    - Strengthen process audits
  - Can focus on increasing the mean (e.g., increase effectiveness, efficiency, etc.)
    - Train people
    - Create checklists
    - Reduce waste and re-work
    - Replicate best practices from other projects
- Can do both
DMAIC - Improve

- Purpose is to develop, implement and evaluate solutions targeted at identified root causes

- Key products
  - Candidate solutions
  - Pilot results
  - Risk assessment
  - Implementation plan

![DMAIC Diagram]

1. DEFINE
2. MEASURE
3. ANALYZE
4. IMPROVE
5. CONTROL

Solutions
FMEA
Pilot
Implementation

NORTHROP GRUMMAN
DMAIC - Control

- **Purpose**: is to make sure problem stays fixed and new methods can be further improved over time

- **Products**
  - Control plan
  - Process documentation
  - Key learnings
The Control Plan

- Systematic tool for identifying and correcting root causes of out-of-control conditions
- Focus is on prevention -- not a “triggering” system
- Questions that must be answered
  - Who owns the process?
  - How will we transition responsibility from the Black Belt to the process owner?
  - How do we ensure we maintain the gains?
  - How do we track our results (performance and financial)?
Institutionalize Key Learnings

- Capture knowledge gained from Six Sigma project
  - Results
  - Key learnings
  - Potential future projects
- Communicate to rest of organization for knowledge sharing and transfer
- Archive in knowledge management repositories
Lessons Learned
Mission Success Requires Multiple Approaches

- Risk Management
- Systems Engineering
- Independent Reviews
- Training, Tools, & Templates

Program Effectiveness
Mission Assurance
Operations Effectiveness

Dashboards for Enterprise-Wide Measurement
Communications & Best-Practice Sharing
Robust Governance Model (Policies, Processes, Procedures)

CMMI Level 5 for Software, Systems, and Services
ISO 9001 and AS-9100 Certification
Six Sigma
Benefits

Based on 18 Northrop Grumman CMMI Level 5 organizations

- Having multiple improvement initiatives helps encourage a change in behavior as opposed to “achieving a level”
  - Reinforces that change (improvement) is a way of life

- The real ROI comes in institutionalizing local improvements across the wider organization
  - CMMI establishes the needed mechanisms

- CMMI and Six Sigma compliment each other
  - CMMI can yield behaviors without benefit
  - Six Sigma improvements based solely on data may miss innovative improvements (assumes a local optimum)

- Training over half the staff has resulted in a change of language and culture
  - Voice of Customer, data-driven decisions, causal analysis, etc.
  - Better to understand and use the tools in everyday work than to adopt the “religion”
Copyright 2005 Northrop Grumman Corporation

Contact Information

Rick Hefner, Ph.D.
Director, Process Management

Northrop Grumman Corporation
One Space Park
Redondo Beach, CA 90278

(310) 812-7290
rick.hefner@ngc.com
Introduction
OMG Systems Modeling Language
(OMG SysML™)
and OOSEM Tutorial

By
Abe Meilich, Ph.D.
abraham.w.meilich@lmco.com

Acknowledged
Original Authors:
Sanford Friedenthal
Alan Moore
Rick Steiner

National Defense Industrial Association
9th Annual Systems Engineering Conference
San Diego, CA
October 23, 2006

Copyright © 2006 by Object Management Group.
Published and used by INCOSE and affiliated societies with permission.
Caveat

• These materials have been modified slightly from the original Tutorial given at INCOSE 2006
  – Softcopy of Full Tutorial available at:

• This material is based on version 1.0 of the SysML specification (ad-06-03-01)
  – Adopted by OMG in May ’06
  – Going through finalization process

• OMG SysML Website
  – http://www.omgsysml.org/
Objectives & Intended Audience

At the end of this tutorial, you should understand the:

• Benefits of model driven approaches to systems engineering
• Types of SysML diagrams and their basic constructs
• Cross-cutting principles for relating elements across diagrams
• Relationship between SysML and other Standards
• Introduction to principles of a OO System Engineering Method

This course is not intended to make you a systems modeler!
You must use the language.

Intended Audience:

• Practicing Systems Engineers interested in system modeling
  – Already familiar with system modeling & tools, or
  – Want to learn about systems modeling
• Software Engineers who want to express systems concepts
• Familiarity with UML is not required, but it will help
Topics

• Motivation & Background
• Diagram Overview
• SysML Modeling as Part of SE Process
• OOSEM – Enhanced Security System Example
• SysML in a Standards Framework
• Transitioning to SysML
• Summary
Background
System Modeling

Integrated System Model Must Address Multiple Aspects of a System
Model Based Systems Engineering
Benefits

• Improved communications
• Assists in managing complex system development
  – Separation of concerns
  – Hierarchical modeling
  – Facilitates impact analysis of requirements and design changes
  – Supports incremental development & evolutionary acquisition
• Improved design quality
  – Reduced errors and ambiguity
  – More complete representation
• Early and on-going verification & validation to reduce risk
• Other life cycle support (e.g., training)
• Enhanced knowledge capture
Modeling at Multiple Levels of the System

Operational Models

System Models

Component Models

Component Models
What is SysML?

• A graphical modelling language in response to the UML for Systems Engineering RFP developed by the OMG, INCOSE, and AP233
  – a UML Profile that represents a subset of UML 2 with extensions

• Supports the specification, analysis, design, verification, and validation of systems that include hardware, software, data, personnel, procedures, and facilities

• Supports model and data interchange via XMI and the evolving AP233 standard (in-process)
What is SysML (cont.)

- *Is* a visual modeling language that provides
  - Semantics = meaning
  - Notation = representation of meaning
- *Is not* a methodology or a tool
  - SysML is methodology and tool independent
UML/SysML Status

• UML V2.0
  – Updated version of UML that offers significant capability for systems engineering over previous versions
  – Finalized in 2005 (formal/05-07-04)

• UML for Systems Engineering (SE) RFP
  – Established the requirements for a system modeling language
  – Issued by the OMG in March 2003

• SysML
  – Industry Response to the UML for SE RFP
  – Addresses most of the requirements in the RFP
  – Version 1.0 adopted by OMG in May ’06 / In finalization
  – Being implemented by multiple tool vendors
Diagram Overview
Relationship Between SysML and UML

UML 2

SysML

UML not required by SysML (UML-UML4SysML)

UML reused by SysML (UML4SysML)

SysML extensions to UML (SysML Profile)
SysML Diagram Taxonomy
4 Pillars of SysML – ABS Example

1. Structure

**bdd [package] VehicleStructure [ABS-Block Definition Diagram]**

```
<block> Library::Electronic Processor
<block> Anti-Lock Controller
<block> Traction Detector
<block> Library::Electro-Hydraulic Valve
```

**ibd [block] Anti-LockController [Internal Block Diagram]**

```
d1:Traction Detector
m1:Brake Modulator
```

**req [package] VehicleSpecifications [Requirements Diagram - Braking Requirements]**

```
Vehicle System Specification
 Braking Subsystem Specification
```

```
<requirement> StoppingDistance
 id="102"
text="The vehicle shall stop from 60 mph within 150 ft on a clean dry surface."
```

```
<requirement> Anti-LockPerformance
 id="337"
text="Braking subsystem shall prevent wheel lockup under all braking conditions."
```

```
<deriveReqt>
```

2. Behavior

**sd ABS_ActivationSequence [Sequence Diagram]**

```
stm TireTraction [State Diagram]
```

```
act PreventLockup [Activity Diagram]
```

3. Requirements

**11 July 2006 Copyright © 2006 by Object Management Group.**

```
4 Pillars of SysML – ABS Example
```

4. Parametrics
Cross Connecting Model Elements

1. Structure

- Anti-LockController
  - Satisfies requirement Anti-Lock Performance
  - Allocate From Activity Detect Loss Of Traction
  - Value DutyCycle: Percentage

2. Behavior

- PreventLockup [Swimlane Diagram]
  - Allocate Anti-LockController
  - Detect Loss Of Traction
  - Traction Loss: Modulate Braking Force
  - Allocate To Braking Subsystem Interface

3. Requirements

- Stopping Distance
  - Requirement
  - Satisfied By Activity Minimum Stopping Distance
  - Verifies Requirement Anti-Lock Performance

4. Parametrics

- StraightLineVehicleDynamics [Parametric Diagram]
  - Distance Equation
  - Velocity Equation
  - Acceleration Equation
  - Friction

11 July 2006

Copyright © 2006 by Object Management Group.
Structural Diagrams

SysML Diagram

Behavior Diagram

Requirement Diagram

Structure Diagram

Activity Diagram

Sequence Diagram

State Machine Diagram

Use Case Diagram

Block Definition Diagram

Internal Block Diagram

Package Diagram

Parametric Diagram

Same as UML 2

Modified from UML 2

New diagram type
Package Diagram

- Package diagram is used to organize the model
  - Groups model elements into a name space
  - Often represented in tool browser
- Model can be organized in multiple ways
  - By System hierarchy (e.g., enterprise, system, component)
  - By domain (e.g., requirements, use cases, behavior)
  - Use viewpoints to augment model organization
- Import relationship reduces need for fully qualified name (package1::class1)
Package Diagram
Organizing the Model

pkg SampleModel [by diagram type]
- Use Cases
- Requirements
- Behavior
- Structure
- EngrAnalysis

pkg SampleModel [by level]
- Enterprise
- System
- Logical Design
- Allocated Design
- Verification

pkg SampleModel [by IPT]
- Architecture Team
- Requirements Team
- IPT A
- IPT B
- IPT C
Package Diagram - Views

- Model is organized in one hierarchy
- Viewpoints can provide insight into the model using another principle
  - E.g., analysis view that spans multiple levels of hierarchy
  - Can specify diagram usages, constraints, and filtering rules
  - Consistent with IEEE 1471 definitions
Blocks are Basic Structural Elements

- Provides a unifying concept to describe the structure of an element or system
  - Hardware
  - Software
  - Data
  - Procedure
  - Facility
  - Person

- Multiple compartments can describe the block characteristics
  - Properties (parts, references, values)
  - Operations
  - Constraints
  - Allocations to the block (e.g. activities)
  - Requirements the block satisfies

<table>
<thead>
<tr>
<th>«block»</th>
<th>BrakeModulator</th>
</tr>
</thead>
<tbody>
<tr>
<td>allocatedFrom</td>
<td></td>
</tr>
<tr>
<td>«activity»Modulate</td>
<td></td>
</tr>
<tr>
<td>BrakingForce</td>
<td></td>
</tr>
<tr>
<td>values</td>
<td></td>
</tr>
<tr>
<td>DutyCycle: Percentage</td>
<td></td>
</tr>
</tbody>
</table>
Block Property Types

- Property is a structural feature of a block
  - **Part property** aka. part (typed by a block)
    - Usage of a block in the context of the enclosing block
    - Example - right-front:wheel
  - **Reference property** (typed by a block)
    - A part that is not owned by the enclosing block (not composition)
    - Example - logical interface between 2 parts
  - **Value property** (typed by value type)
    - Defines a value with units, dimensions, and probability distribution
    - Example
      - Non-distributed value: tirePressure:psi=30
      - Distributed value: «uniform» {min=28,max=32} tirePressure:psi
Using Blocks

• Based on UML Class from UML Composite Structure
  – Eliminates association classes, etc.
  – Differentiates value properties from part properties, add nested connector ends, etc.
• Block definition diagram describes the relationship among blocks (e.g., composition, association, classification)
• Internal block diagram describes the internal structure of a block in terms of its properties and connectors
• Behavior can be allocated to blocks
Block Definition vs. Usage

**Definition**
- Block is a definition/type
- Captures properties, etc.
- Reused in multiple contexts

**Usage**
- Part is the usage in a particular context
- Typed by a block
- Also known as a role
Internal Block Diagram (ibd)
Blocks, Parts, Ports, Connectors & Flows

Enclosing Block

Connector

Reference Property (in, but not of)

Item Flow

Part

Port

Internal Block Diagram Specifies Interconnection of Parts
Reference Property Explained

S1 is a reference part in ibd shown in dashed outline box

ibd [block] Anti-LockController [Internal Block Diagram]

c2: sensor Interface

s1: Sensor

c1: modulator Interface

d1: Traction Detector

m1: Brake Modulator

bdd [package] VehicleStructure
SysML Port

- Specifies interaction points on blocks and parts
  - Supports integration of behavior and structure
- Port types
  - Standard (UML) Port
    - Specifies a set of operations and/or signals
    - Typed by a UML interface
  - Flow Port
    - Specifies what can flow in or out of block/part
    - Typed by a flow specification

2 Port Types Support Different Interface Concepts
Port Notation

- **Standard Port**
  - Provided interface: (provides the operations)
  - Required interface: (calls the operations)

- **Flow Port**
  - Item flow
  - part1:
  - part2:
Parametrics

- Used to express constraints (equations) between value properties
  - Provides support for engineering analysis (e.g., performance, reliability)
- Constraint block captures equations
  - Expression language can be formal (e.g., MathML, OCL) or informal
  - Computational engine is defined by applicable analysis tool and not by SysML
- Parametric diagram represents the usage of the constraints in an analysis context
  - Binding of constraint usage to value properties of blocks (e.g., vehicle mass bound to \( F = m \times a \))
Defining Vehicle Dynamics

Defining Reusable Equations for Parametrics
Vehicle Dynamics Analysis

Using the Equations in a Parametric Diagram to Constrain Value Properties

par [constraintBlock] StraightLineVehicleDynamics [Parametric Diagram]

- \( v.chassis.tire.\) Friction:
- \( v.brake.abs.m1.\) DutyCycle:
- \( v.brake.rotor.\) BrakingForce:
- \( v.Weight.\)

\[ f = (t_f^*b_f)^*(1-t_l) \]

\[ F = m^*a \]

\[ v = dx/dt \]

\[ a = dv/dt \]
Behavioral Diagrams

- SysML Diagram
- Behavior Diagram
  - Activity Diagram
  - Sequence Diagram
  - State Machine Diagram
  - Use Case Diagram
- Requirement Diagram
- Structure Diagram
  - Block Definition Diagram
  - Internal Block Diagram
  - Package Diagram

- Parametric Diagram

Legend:
- Same as UML 2
- Modified from UML 2
- New diagram type
Activities

- Activity used to specify the flow of inputs/outputs and control, including sequence and conditions for coordinating activities
- Secondary constructs show responsibilities for the activities using swim lanes
- SysML extensions to Activities
  - Support for continuous flow modeling
  - Alignment of activities with Enhanced Functional Flow Block Diagram (EFFBD)
Activity Diagram Notation

Activity Notation:
- Activity: MonitorTraction
- Action: Calculate Traction
- Decision: [loss of traction] else Calculate Modulation Frequency
- Object Flow: Modulation Frequency
- Flow Final Node: SpeedInput, WheelRevs
- Activity Parameter Node: Angular Velocity, Speed
- Fork: Calculate Wheel Velocity, Calculate Car Velocity
- Initial Node: act
- Pin

Notes:
- Join and Merge symbols not included
- Activity Parameter Nodes on frame boundary correspond to activity parameters

11 July 2006
Copyright © 2006 by Object Management Group.
Activity Diagrams
Pin vs. Object Node Notation

- Pins are kinds of Object Nodes
  - Used to specify inputs and outputs of actions
  - Typed by a block or value type
  - Object flows connect object nodes

- Object flows between pins have two diagrammatic forms
  - Pins shown with object flow between them
  - Pins elided and object node shown with flow arrows in and out

Pins must have same characteristics (name, type etc.)
Explicit Allocation of Behavior to Structure Using Swimlanes

Activity Diagram (without Swimlanes)

Activity Diagram (with Swimlanes)

11 July 2006
Copyright © 2006 by Object Management Group.
SysML EFFBD Profile

EFFBD - Enhanced Functional Flow Block Diagram

Aligning SysML with Classical Systems Engineering Techniques
Distill Water Activity Diagram
(Continuous Flow Modeling)

Representing Distiller Example in SysML
Using Continuous Flow Modeling
Activity Decomposition

**Definition**

`bdd PreventLockup [Activity Breakdown]`

- «activity» PreventLockup
  - a1: DetectLossOf Traction
  - a2: ModulateBrakingForce

**Use**

`act PreventLockup [Activity Diagram]`

- a1: DetectLossOf Traction
- a2: ModulateBrakingForce

Traction Loss:

11 July 2006 Copyright © 2006 by Object Management Group.
Interactions

- Sequence diagrams provide representations of message based behavior
  - represent flow of control
  - describe interactions
- Sequence diagrams provide mechanisms for representing complex scenarios
  - reference sequences
  - control logic
  - lifeline decomposition
- SysML does not include timing, interaction overview, and communications diagram
Black Box Interaction (Drive)

UML 2 Sequence Diagram Scales by Supporting Control Logic and Reference Sequences
Black Box Sequence (StartVehicle)

Simple Black Box Interaction

References Lifeline Decomposition For White Box Interaction
State Machines

- Typically used to represent the life cycle of a block
- Support event-based behavior (generally asynchronous)
  - Transition with trigger, guard, action
  - State with entry, exit, and do-activity
  - Can include nested sequential or concurrent states
  - Can send/receive signals to communicate between blocks during state transitions, etc.
Operational States (Drive)

Transition notation: trigger[guard]/action
Use Cases

- Provide means for describing basic functionality in terms of usages/goals of the system by actors
- Common functionality can be factored out via include and extend relationships
- Generally elaborated via other behavioral representations to describe detailed scenarios
- No change to UML
Operational Use Cases
Cross-cutting Constructs

- Allocations
- Requirements

SysML Diagram

- Behavior Diagram
- Requirement Diagram
- Structure Diagram

Activity Diagram  Sequence Diagram  State Machine Diagram  Use Case Diagram  Block Definition Diagram  Internal Block Diagram  Package Diagram

- Same as UML 2
- Modified from UML 2
- New diagram type
Allocations

- Represent general relationships that map one model element to another
- Different types of allocation are:
  - Behavioral (i.e., function to component)
  - Structural (i.e., logical to physical)
  - Software to Hardware
  - ....
- Explicit allocation of activities to structure via swim lanes (i.e., activity partitions)
- Both graphical and tabular representations are specified
Different Allocation Representations
(Tabular Representation Not Shown)

Allocate Relationship

Explicit Allocation of Activity to Swim Lane

Compartiment Notation

Callout Notation
SysML Allocation of SW to HW

- In UML the deployment diagram is used to deploy artifacts to nodes
- In SysML allocation on ibd and bdd is used to deploy software/data to hardware
Requirements

• The «requirement» stereotype represents a text based requirement
  – Includes id and text properties
  – Can add user defined properties such as verification method
  – Can add user defined requirements categories
    (e.g., functional, interface, performance)

• Requirements hierarchy describes requirements contained in a specification

• Requirements relationships include DeriveReqt, Satisfy, Verify, Refine, Trace, Copy
Requirements Breakdown

**req [package] HSUVRequirements [HSUV Specification]**

- **HSUVSpecification**
  - **Eco-Friendliness**
  - **Performance**
    - **Braking**
    - **FuelEconomy**
    - **Accelleration**
  - **Emissions**
    - Id = "R1.2.1"
    - text = "The vehicle shall meet Ultra-Low Emissions Vehicle standards."
  - **RefinedBy**
    - «useCase» HSUVUseCases::Accelerate
  - **VerifiedBy**
    - «testCase» MaxAcceleration
  - **SatisfiedBy**
    - «block» PowerSubsystem

**Requirement Relationships Model the Content of a Specification**

11 July 2006 Copyright © 2006 by Object Management Group.
Example of Derive/Satisfy Requirement Dependencies

Client depends on supplier (i.e., a change in supplier results in a change in client)

Arrow Direction Opposite Typical Requirements Flow-Down
Problem and Rationale can be attached to any Model Element to capture issues and decisions.
SysML Modeling
as Part of the SE Process
OOSEM – Enhance Security System (ESS) Example
System Development Process

Integrated Product Development (IPD) is essential to improve communications.

A Recursive V process that can be applied to multiple levels of the system hierarchy.

Stakeholder Reqs

Manage System Development

Plan
Systems Modeling Activities - OOSEM

Major SE Development Activities

- Define System Requirements
  - Mission use cases/ scenarios
  - System use cases/ scenarios
  - Elaborated context
  - Req’ts diagram

- Define Logical Architecture
  - System use cases/ scenarios
  - Elaborated context
  - Req’ts diagram
  - Logical architecture

- Validate & Verify System
  - Test cases/ procedures

- Synthesize Physical Architecture
  - Node diagram
  - HW, SW, Data architecture

- Optimize & Evaluate Alternatives
  - Engr Analysis Models
  - Trade studies

Common Subactivities

Enhanced Security System Example

• The Enhanced Security System is the example for the OOSEM material
  – Problem fragments used to demonstrate principles
  – Utilizes Artisan RTS™ Tool for the SysML artifacts
Market Needs

id# = SS1

ESS System Specification

id# = SS102

txt = System shall detect intruder entry and exit ...

IntruderDetection

id# = SS111

R111

satisfiedBy Entry/Exit Subsystem

verifiedBy Entry/Exit Detection Test

ESS Logical Requirements

id# = LR1

ESS Logical Design Models

ESS Allocated Requirements

id# = AR1

ESS Allocated Design Models

ESS Enterprise Models

ESS System Models
Operational View Depiction

bdd [package] Enterprise (As Is)

Central Monitoring Station As-Is

Comm Network

Residence

Dispatcher

Police

Intruder
ESS Enterprise As-Is Model
ESS Operational Enterprise To-Be Model

bdd [package] ESS Enterprise (To Be)

Domain To-Be

Protected Site

- Intruder
  - Customer

Physical Environment

- Single-family Residence
- Multi-family Residence
- Business

ESS

- ESS Operational Enterprise
  - OperationalAvailability = {>.99}
  - MissionResponseTime = {<5 min}
  - OperationalCost = {TBD}
  - CostEffectiveness

MonitorSite ()
DispatchEmergencyServices ()
ProvideEmergencyResponse ()

ESS Operational Enterprise

- Emergency Services
  - Assess Report ()
  - Report Update ()
  - Dispatch Police ()

Dispatcher
Responder
Fire
Paramedic
Police

System Use Cases - Operate

- Operate
- Monitor Site
- Activate/Deactivate
- Respond to Break-In
- Respond to Fire
- Respond to Medical
- Respond

UC [package] System Use Cases

«include»

«extend»
System Scenario: Activity Diagram
Monitor Site (Break-In)
ESS Logical Design – Example Subsystem

```
ibd [subsystem]Entry/Exit Subsystem

: Door Input
: Window Input
: Door Input
: Window Input

«logical» : Entry Sensor
m+n : SensedEntry

«logical» : Entry/Exit Monitor

«logical» : Exit Sensor
m+n : SensedExit

: Entry/Exit Alert Status

«logical» : Event Monitor

«store» : Event Log

: Alert Status
```
### ESS Allocation Table (partial)

- Allocating Logical Components to HW, SW, Data, and Procedures components

<table>
<thead>
<tr>
<th>Logical Components</th>
<th>Entry Sensor</th>
<th>Exit Sensor</th>
<th>Perimeter Sensor</th>
<th>Entry/Exit Monitor</th>
<th>Event Monitor</th>
<th>Site Comms I/F</th>
<th>Event Log</th>
<th>Customer I/F</th>
<th>Customer Output Mgr</th>
<th>System Status</th>
<th>Fault Mgr</th>
<th>Alarm Generator</th>
<th>Alarm I/F</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Physical Components</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>«hardware»</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Optical Sensor</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DSL Modem</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>User Console</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Video Camera</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Alarm</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>**Software»</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Device Mgr</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>SF Comm I/F</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>User I/F</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Event Mgr</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Site Status Mgr</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Site RDBMS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMS RDBMS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>**Data»</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Video File</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMS Database</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Site Database</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>**«hardware»</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

ESS Parametric Diagram
To Support Trade-off Analysis

```
par [block] EnterpriseEffectivenessModel

«moe» MissionResponseTime

«moe» OperationalAvailability

of1 : ObjectiveFunction

MRT
OA
OC

{CE= Sum(w1*u(OA)+w2*u(MRT)+w3*u(OC))}

CE

«moe» OperationalCost

«moe» CostEffectiveness
```
Entry/Exit Test Case

**sd** Entry/Exit Detection Test

**Description**

«testComponent» : IntruderEmulator

«sut» «hardware»

Door[1]

:/ Optical Sensor

«sut» «hardware»

Window[4]

:/ Optical Sensor

«sut» «hardware»

Site Processor

«sut» «hardware»

DSL Modem

---

seq

Intruder enters through front door

Door sensor detects entry

New alert status sent to central system

Intruder leaves through lounge window

Window sensor detects exit

Changed alert status sent to central system

seq

Enter

:SensedEntry

IntruderEntry : Alert Status

Exit

:SensedExit

Intruder Exit : Alert Status

---

SysML in a Standards Framework
Systems Engineering Standards Framework (Partial List)

- **Process Standards**
  - EIA 632
  - ISO 15288
  - IEEE 1220
  - CMMI

- **Architecture Frameworks**
  - FEAF
  - DoDAF
  - MODAF
  - Zachman FW

- **Modeling Methods**
  - HP
  - OOSE
  - SADT
  - Other

- **Modeling & Simulation Standards**
  - IDEF0
  - SysML
  - UPDM
  - HLA
  - MathML

- **Interchange & Metamodelling Standards**
  - MOF
  - XMI
  - STEP/AP233

- **Implemented By Tools**

- **Data Repository**

11 July 2006

Copyright © 2006 by Object Management Group.
Standards-based Tool Integration with SysML

Systems Modeling Tool

Other SE Engineering Tools

Model/Data Interchange

AP233/XMI

AP233/XMI
Participating SysML Tool Vendors

- Artisan
- EmbeddedPlus
  - 3rd party IBM vendor
- Sparx Systems
- Telelogic (includes I-Logix)
- Vitech

**Note:** Free Visio SysML Template available at OMG SysML site ([http://www.omgsysml.org](http://www.omgsysml.org))
UML Profile for DoDAF/MODAF (UPDM) Standardization

- Current initiative underway to develop standard profile for representing DODAF and MODAF products
  - Requirements for profile issued Sept 05
  - Final submissions expected Dec ‘06
- Multiple vendors and users participating
- Should leverage SysML
Transitioning to SysML
Using Process Improvement To Transition to SysML

- Assess & Measure Improvement
- Plan Improvement
- Define Improvement
- Continuous Improvement Cycle
- Deploy Improvement
- Pilot Improvement
**Integrated Tool Environment**

<table>
<thead>
<tr>
<th>CM/DM</th>
<th>Requirements Management</th>
<th>Engineering Performance Analysis</th>
<th>System Modeling</th>
<th>Software Modeling</th>
<th>Hardware Modeling</th>
<th>Verification &amp; Validation</th>
<th>Specialty Engineering Analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product Data Management</td>
<td></td>
<td>SoS / DoDAF / Business Process Modeling</td>
<td>SysML</td>
<td>UML 2</td>
<td>VHDL, CAD, ..</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

11 July 2006  Copyright © 2006 by Object Management Group.
Summary and Wrap up
Summary

- SysML sponsored by INCOSE/OMG with broad industry and vendor participation
- SysML provides a general purpose modeling language to support specification, analysis, design and verification of complex systems
  - Subset of UML 2 with extensions
  - 4 Pillars of SysML include modeling of requirements, behavior, structure, and parametrics
- OMG SysML Adopted in May 2006
- Multiple vendor implementations announced
- Standards based modeling approach for SE expected to improve communications, tool interoperability, and design quality
References

- OMG SysML website
  - http://www.omgsysml.org
- UML for Systems Engineering RFP
  - OMG doc# ad/03-03-41
- UML 2 Superstructure
  - OMG doc# formal/05-07-04
- UML 2 Infrastructure
  - OMG doc# ptc/04-10-14
Systems Engineering for the Joint Capabilities Integration and Development System (JCIDS)

Tutorial for the 9th NDIA Systems Engineering Conference
Agenda and Presenters

- Introduction to JCIDS – Chris Ryder
- Applying the Systems Engineering Method for JCIDS – Dave Flanigan
- Model-Driven Systems Engineering for JCIDS – Jennifer Rainey
- JCIDS Functional Analyses – Dave Krueger and Chris Ryder
- Summary – Chris Ryder
Purpose of the Tutorial

- JCIDS prescribes a joint forces approach to identify capability gaps against current force capability needs
- The Systems Engineering (SE) Method applies to each iteration of the systems life-cycle from capability inception through system retirement
- Good systems engineering practice is necessary for successfully implementing JCIDS
- JCIDS Functional Analyses perform critical problem solving activities
- Use of model-driven SE facilitates JCIDS throughout the systems life-cycle
What is JCIDS?

- Capabilities-based assessment (CBA) composed of a structured
- Four-step methodology that defines capability gaps
- Capability needs and approaches to provide those capabilities within a specified functional or operational area.
JCIDS Is an Engineering Intensive Function

- JCIDS activities are fundamental Systems Engineering actions
  - Consistent with the Systems Engineering Method
  - Performed at early concept analysis and development
  - But also at each capability upgrade
- JCIDS analysis quantifies material and non-material options
  - Systems Engineering life-cycle phases quantifies the phases of “Materialization”
    - Abstract concepts in early phases
    - Concrete systems and subsystems as the life cycle progresses
 JCIDS Process

DOD Strategic Guidance

Joint Operations Concepts

Joint Operating Concepts
Joint Functional Concepts
Joint Integrating Concepts

Functional Area Analysis

Functional Needs Analysis

JCD

DOTLPF Analysis (Non-materiel Approaches)

Ideas for Materiel Approaches

Analysis of Materiel/Non-materiel Approaches

Alternative N
Alternative 2
Alternative 1

Integrated Architectures

CPD

CDD

ICD

DCR

Functional Solution Analysis

Post Independent Analysis

*JCJSN 3170.01B
JCIDS Events

- Functional Area Analysis (FAA)
  - Identify operational task, conditions, and standards needed to accomplish military objectives
  - **Result:** Tasks to be accomplished
- Functional Needs Analysis (FNA)
  - Assess ability of current and programmed capabilities to accomplish the tasks
  - **Result:** List of capability gaps
- Functional Solutions Analysis (FSA)
  - Operational based assessment of DOTMLPF approaches to solving capability gaps
  - **Result:** Potential DOTMLPF approaches to capability gaps
- Post Independent Analysis
  - Independent analysis of approaches to determine best fit
  - **Result:** Initial Capabilities Document
JCIDS

- JCIDS analytical process stresses the fundamentals for applying an effective systems engineering program by any accepted standard
- It guides the “front-end” phases of the SE process for each capability iteration
  - Enterprise (operational) analysis
  - Requirements definition
  - Life-cycle phase
- The analysts must have a thorough understanding of existing capabilities as well as the capability needs
- The JCIDS analysis team eventually determines the optimum combination of material and non-material alternatives to achieve the capability needs to the Battle Force
Perspective

- Not an authoritative review of DoD policy and procedures
  - Chairman of the Joint Chiefs of Staff Instruction (CJCSI) 3170.01, “Joint Capabilities Integration And Development System”
  - Chairman of the Joint Chiefs of Staff Manual (CJCSM) 3170.01, “Operation of the Joint Capabilities Integration and Development System”

- Relationship to the acquisition process

- Discuss some thoughts on implementation
  - FAA, FNA, FSA, and PIA
  - No definitive cookbook for implementation
The JCIDS Meta-Model

- OMG SysML™ model of JCIDS activities and artifacts
  - Activities
    - FAA
    - FNA
    - FSA
    - PIA
  - Artifacts
    - Architecture model
      - Including capability use cases
    - ICD, CDD, CPD
    - DOTMLPF Change Recommendations
    - Integrated Threat Warning Assessment (ITWA)
    - CONOPS
OMG SysML™

- Diagrams used throughout this presentation are constructed using the Object Management Group’s SysML (Systems Modeling Language)
- Systems engineering extension to Unified Modeling Language (UML™) 2.0
- OMG SysML is a standardized family of diagrams depicting system elements, their behaviors and their relationships with other elements internal and external to the system
  - Captures operational and systems requirements
  - Documents element parametrics and constraints
  - Methodology independent

For a detailed discussion on SysML, each of you are invited to attend Abe Meilich’s SysML Tutorial this afternoon
Targeting and Bomb Damage Assessment

- Read the attached “Statement of Operational Need” at the break
- TBDA represents a required capability to:
  - Maintain persistent coverage over a target area
  - Acquire fire-control quality track files on moving targets
  - Provide the means to determine whether a target was sufficiently destroyed or neutralized
  - Must be able to be deployed by a small team of ground-based personnel
  - Controlled by the local ground commander

TBDA is a FICTICIOUS Case! Any similarity with any capability or system, real or imagined, is purely coincidental!
TBDA Presentation

- This case presents a very limited sample of artifacts and elements that are part of the FAA, FNA and FSA SE Model
- Intent is to illustrate modeling possibilities using SysML
  - Requirements traceability
  - Entities with their behaviors and relationships
    - Material and non-material
  - Standards that govern architectural elements
Systems Engineering Method

David Flanigan

October 23, 2006
Discussion Topics

- Describe Systems Engineering Method (SEM) in JCIDS context
  - Identify and describe four “root” steps
  - Identify inputs/conditions for each step
  - Identify outputs/products from each step
  - Inter-relationships among the steps
- Show linkage to JCIDS and the Systems Engineering lifecycle
Systems Engineering Method

- Regardless of the analytical phase performed by the JCIDS SE team,
  - The basic application of the SE method is constant throughout the process
- Each SE Method activity is performed in some form or to some degree in each phase of the system life-cycle
Systems Engineering Method

Measures of Effectiveness

“Problem Space”

“Solution Space”

Measures of Performance

Systems Engineering Method

Adapted from Kossiakoff & Sweet “Systems Engineering Principles and Practice”

Crossing the boundary from the problem space to the solution space costs $$
The Systems Engineering “V” is iterative and tailored throughout the entire lifecycle.
Systems Engineering Method Over Life Cycle

The generic SE method can be mapped to each specific iteration of the SE “V”
The front end of JCIDS has no Systems Engineering “V”
Need to do good SE at the start of the lifecycle, since each phase relies on products from the previous phase.

JCIDS is performed over the entire lifecycle.
Systems Engineering Method

- **Need**
  - Requirements (Capability) Analysis
  - Functional Definition
  - Physical Definition
  - Functional Area Analysis
  - Functional Needs Analysis
  - Functional Solutions Analysis
  - Potential Solutions
  - Design Validation
  - Program Independent Assessment

Ensure the traceability of needs and requirements are present through the entire lifecycle.
Systems Engineering Method
Phase 1: Requirements (Capability) Analysis

Requirements (Capability) Analysis

- Problem Definition
- Rationale, Scope, & Context
- Sponsor-derived Problem Set
- Legacy Operational Activities
- Capabilities

To Functional Analysis Phase
Problem Definition

- At one point in time there is a problem that must be solved due to:
  - Deficient capability with existing systems
  - Desire to improve existing performance
- Need to understand what the objectives are to provide the desired capability
- Define the operational context within the Capability Enterprise!

Did we define the problem correctly?
Did we define the correct problem???
Example Requirements Analysis Products

- Clear(er) definition of the problem
- Proper scope of the problem
- Operational context documents and data bases
  - Design Reference Mission
  - Strategy-to-Task Mapping
  - Concept of Operations
  - Physical Environment Database
  - Threat Representation Database
  - Blue Capabilities Database
- Relevant Operational Views

Captured within a SE Requirements Model
Systems Engineering Method
Phase 2: Functional Definition

From Requirements Definition

- Functional Improvements
- Directed Functions
- Functional Decomposition
- Technological Contributions

To Physical Definition

Functional Definition
Typical Functional Definition Products

- Functional Decomposition of required activities
  - Functional diagrams (Functional Flow Block Diagrams, UML Activity Diagrams)
- Associated metrics with these functions (threshold / objective)
- Analysis process that determines if you can solve with a material / non-material / both solution
  - Be able to document and defend this process
- How do we know it’s right?
  - The functions are legitimate, correct, and validated by users
- Functional Area Analysis
- Relevant operational views

Functional Analysis Documented in a SE Functional or Physical Model
Systems Engineering Method
Phase 3: Physical Definition

From Functional Definition

Material / Non-Material solution?

Non-Material

Material

DOTMLPF Elements

Collect Candidate Systems

Physical Definition

To Design Validation
Typical Physical Definition Products

- Provide system alternatives towards satisfying required functionality
  - Assignment of functions to physical elements
- DOTMLPF analysis products
  - Based on the functional definition phase
- CONOPS changes / recommendations
  - Based on DOTMLPF analysis
- Risk management strategies of the system
- System roadmaps to bridge the gap between the current and future capabilities
- Functional Needs Analysis
- Relevant operational and SYSTEMS views

SE Physical Model with Physical Definition Begins
Evolution Toward a Systems Model
Systems Engineering Method
Phase 4: Design Validation

From Physical Definition

Analysis

M&S

Problem satisfied?

Yes

No

Reassess requirements, functional elements or physical details

To next life-cycle phase: Requirements Definition

Design Validation
Typical Design Validation Products

- Demonstrate the analysis documents the assumptions, follows a rigorous process, and arrives at meaningful conclusions that are justifiable
  - There may be multiple processes and products dependent on the sponsor, personnel/time availability, experience
  - This may be an iterative process for ICD, CDD, CPD
- Trade studies
- VV&A
- Risk Management
- Cost Analysis
- Force Allocation
- Functional Solutions Analysis
- Program Independent Assessment

Attain a Fully Validated Systems Engineering Model
Systems Engineering Methodology
Linkage to JCIDS Summary

- SE methods can be used to produce JCIDS products/artifacts
- SE methods can iterate throughout the DoD 5000 lifecycle
- Good SE methods can produce JCIDS
- Bad SE methods can produce JCIDS
- Producing JCIDS does not guarantee good SE

Good SE  ↔  Effective JCIDS
Applying Model-Driven Systems Engineering Practices to JCIDS

Jennifer Rainey

October 23, 2006
Discussion Topics

- Purpose – a model-driven systems engineering (MDSE) approach supports the JCIDS lifecycle process

- What is Model-Driven Systems Engineering?

- How to Apply Model-driven Systems Engineering for JCIDS?
Purpose

- A Model Driven Systems Engineering (MDSE) approach supports the entire defense acquisition, technology, and logistics lifecycle
  - A systems engineering model provides traceability from system development back to initial JCIDS process and war fighting need
  - MDSE focuses on techniques that drive capability identification
    - Documents entire system lifecycle
    - Identifies the capabilities, capability gaps, and materiel/non-materiel solutions
    - Develops foundation for integrated architectures
  - JCIDS is a concept-centric capabilities identification process
    - “The process to identify capability gaps & potential material and non-materiel solutions must be supported by a robust analytical process that incorporates innovative practices…”
  - CJCSI 3170.01E 11 May 2005

Use of model-driven SE facilitates JCIDS throughout the systems lifecycle
Integrated Defense Acquisition, Technology, & Logistics Lifecycle Management Framework

MDSE should be used here

MDSE is used here
Systems Engineering Model

- A model is a representation of a system
  - Assists stakeholders, including engineers, to understand something that is not easily comprehensible
  - Communicates the organization of the system to stakeholders
  - Enhances understanding of interfaces, relationships, operations and risk
  - Continually updated

- Systems Engineering Model
  - Build as the basis for JCIDS analysis
  - Covers the problem and solution space
  - Contains the objects, relationships and the data
    - Requirements, Functional, and Physical
  - Develops the integrated architecture

Systems Engineering Model is a Living Entity
Model-Driven Systems Engineering

- Establish system model bases on:
  - Requirements model
  - Functional model
  - Physical model

- Show relationships between the models
  - Link operational needs to capabilities
  - Link capabilities to requirements
  - Link requirements to functions
  - Link functions to systems

“If you don’t model it, you won’t understand it.”

Ivar Jacobson
Systems Engineering Method

Building blocks of an integrated architecture

Adapted from Kossiakoff & Sweet "Systems Engineering Principles and Practice"
Requirements Model

- Requirements Analysis
  - Define/scope the problem space
  - Identify “capabilities” during JCIDS process to meet war fighting needs
    - Capabilities turn into requirements later in the lifecycle
  - Analyze capabilities/requirements
    - Assess against “as-is” capabilities/systems, identify gaps
    - Ensure they are necessary, concise, attainable, complete, consistent, unambiguous, and verifiable
    - Create requirements traceability
  - Products:
    - Framework for Operational Views
      - Fulfill need to develop DoDAF operational view artifacts
      - Sets standards to be used needed for technical view artifacts
    - Metrics
      - Measures of Effectiveness
      - Measures of Performance
    - Operational context documents
Functional Model

- **Functional Definition**
  - Implementation free identification of required activities
  - Establish functional decomposition
    - Use Cases, Operational Scenarios
    - Functional Flow Block Diagrams (FFBDs)
    - Unified Modeling Language (UML) Activity Diagrams
  - Can model the time sequencing of the functions
  - Show data or information flow between functions
    - Fulfill need to develop DoDAF operational view artifacts
    - Fulfill need to develop DoDAF system view artifacts

- **Products:**
  - Capabilities/functionality needed to meet requirements
  - Refined performance metrics
  - Framework for Operational and System Views
Physical Model

- Physical Definition – solution space
  - Set the system context or boundary
    - Context diagrams
    - Class diagrams
  - Allocate functions to physical elements
    - Evaluate “to-be” capabilities against “as-is” capabilities and systems to identify the “capability gaps” and “redundancies”
    - Establish link to requirements

- Products:
  - System Elements
  - Relevant system views
  - System data exchanges
  - System roadmaps to bridge the capability gap

Formed the Systems Engineering Model
The Systems Engineering Model

- Where it all ties together!
  - Formed by establishing the relationships between the requirements, functional, and physical elements of the model
    - Requirements (capabilities) link to functions, functions are allocated to physical components
      - Early in the process, the system solution can be expressed as a “black box”
      - As the lifecycle advances, the physical model is further refined into sub-systems
        - Ensure every requirement is linked to a function
        - Ensure every function is allocated to physical element
    - The MDSE process forms the basis for the integrated architecture
  - Supports impact analysis
    - The SE Model developed during upfront JCIDS is the same model used during the entire acquisition lifecycle
    - Traceability is maintained back to the original capability need identified
      - Allows greater understanding of the impact of how changing one element in the model, impacts other areas
MDSE Basis for Integrated Architectures

Functional Area Analysis

- Functional Needs Analysis
- DOTLPF Analysis (Non-materiel Approaches)
- Ideas for Materiel Approaches

DOD Strategic Guidance

- Joint Operations Concepts
- Joint Operating Concepts
- Joint Functional Concepts
- Joint Integrating Concepts

Integrated Architectures

- Alternative N
- Alternative 2
- Alternative 1

Analysis of Materiel/Non-materiel Approaches

- Post Independent Analysis

CPD
- CDD
- ICD
- DCR

Functional Solution Analysis
Architectures in JCIDS

- “Integrated Architectures” are a foundation for the analytical process
  - Stated requirements, attributes, and measures
  - Meets DoDAF needs
  - Used during upfront JCIDS, concept refinement, technology development, system integration, system development, and production
  - System model defines the architecture used during the lifecycle

- “Key components of the CDD and CPD are the integrated architecture products that ensure the DoD understands the linkages between capabilities and systems and can make appropriate acquisition decisions.” CJCSI 3170.01E 11 May 2005

The “Systems Model” becomes the basis for architecture and JCIDS analysis
Architecture Views

- Architecture Views
  - A view is a different “slice” of the model
  - Provides a look “inside” the model
  - Includes information relevant to the stakeholder

- An architecture engenders a multitude of artifacts
  - Most are derived using the same information and data elements
    - Can be obtained from the systems engineering model
  - DoDAF architecture views are specific types of artifacts
    - Includes Operational, Systems, and Technical Views
  - DoDAF architecture views are just a few of the possible model views
A Systems Engineering model captures the essential elements of the systems engineering life-cycle

“Dynamic and recursive process” (Bootch, Rumbaugh, Jacobson)
- Iteratively captures enterprise capabilities and system requirements
- Promotes incorporation of technology evolution

Forms basis for sound, long-term systems engineering and analysis
- Compliant with DoDAF and JCIDS

Model-Driven SE in Defense Systems Acquisition becomes Model-Driven JCIDS
How to Apply MDSE to JCIDS

- Establish a meta-model to understand the framework for the process

- Meta-model is another abstraction, highlighting the properties of a model
  - Explicit description (constructs and rules) of how a domain-specific model is built

- JCIDS meta-model is composed of:
  - Dynamic elements – modeling the behavior over time
  - Logical elements – static view of the objects and classes

Need to model JCIDS process as a “meta-model”
The JCIDS Meta-Model

- **Dynamic Component**
  - Incorporates model-driven analyses within the JCIDS process
  - Standardizes SE modeling methods demonstrate utility for modeling JCIDS capabilities
  - Applies the model-driven approach to each JCIDS analytical phase
    - Leading up to JCIDS analyses documentation
    - Appropriate for capability iterations throughout the Warfare Systems’ lives
    - Easily updated and maintained
    - Use throughout the acquisition lifecycle

- **Logical Component**
  - The Capability Object exists within the “Capability Enterprise”
  - Captures logical and dynamic elements
  - Identifies the attributes and operations of a Capability Object functioning within the operational domain
  - Identifies “Non-Materiel” elements of DOTMLPF
JCIDS Meta-Model Dynamic Component

- FAA, FNA, FSA, and PIA are represented as use cases
  - Each phase represents a dynamic set of activities
  - With post-condition “Result of Value”

- Relates the JCIDS activities to the process of SE/Architecture modeling
  - Understanding the As-Is Enterprise and evolving the To-Be mission scenarios and use cases
JCIDS Dynamic Model

- Perform FAA
- Perform FNA
- Perform FSA
- Conduct PIA

<<Capability>>
Capability Object

Capability Sponsor
JCIDS Meta-Model Logical

- Focus of analysis is on a Capability Object
  - Enables itself within the Capability Enterprise
- Identification of needed capabilities to fulfill war fighting needs (FAA)
- Baseline Capability Enterprise is composed of As-Is capabilities of legacy As-Is Warfare System
- Comparison of To-Be capabilities against the As-Is baseline yields the Capability Gap(s) (FNA)
- Evolve the capability and allocate to physical To-Be Warfare System (FSA)
- DOTMLPF applies needs analysis and potential solutions
Capability Object

- Form of “System Object” as defined by Object Oriented Systems Engineering Methods (OOSEM)
  - Performs operations on behalf of itself and/or other objects
    - Provide output result of value
    - Provide services and information related elements within the domain
  - Possesses measurable properties
    - Physical, data, performance

- Capability Objects, like all UML classes, possess:
  - Attributes
  - Operations
  - Associations
Capability Object and the Warfare System

class Capabilities Enterprise

«capability»
Capability Enterprise

«capability»
Capability Object
+ Attributes:
+ Tasks()

«system»
Warfare System

Component of

+Supports/Affects

«system»
System As-Is

«system»
System To-Be

+Affected/Enhanced by

«capability»
Capability As-Is

«capability»
Capability To-Be

DOTMLPF

+Affected/Enhanced by

DOTMLPF As-Is

DOTMLPF To-Be

NDIA

slide 55
Capability Object

- Attributes:
  - Task(s)

Class Association

- Employed In
- Uses

Scenario

- «capability» Cabability Object
  - Attributes
  - Task(s)

Attribute

- Property Type
- Parameter

Operation

- Evolution
- Function

Measures

- KPP
- Threshold
- Objective
- Effectiveness
- Performance
- Suitability

- FAA
- FNA/FSA
- FSA
- ICD/CDD/CPD
Transition from Capability to System

- Use capability object to perform assessments to satisfy DOTMLPF
  - Analysis of Materiel/non-materiel approaches
  - Analysis of Alternatives
  - Initial Capability Document
  - Investigate if a modification to any element of DOTMLPF except the “M” will enhance the Capability Enterprise
  - A far less expensive option
- DOTMLPF elements can be modeled as classes
  - Each non-materiel element possess attributes and operations
  - Helpful to define a meta-class early in the process to understand element components and relationship
The Legacy System

DOTMLPF can also transition from “As-Is” to “To-Be”
Model-driven Approach Facilitates JCIDS

Building blocks of an integrated architecture

Adapted from Kossiakoff & Sweet “Systems Engineering Principles and Practice”

Model-driven Approach Facilitates JCIDS

Building blocks of an integrated architecture

Adapted from Kossiakoff & Sweet “Systems Engineering Principles and Practice”
Benefits of Model-Driven Approach

- Traceable back to initial FAA and war fighting need
  - Changes to system requirements can be evaluated against the “to-be” capability identified during FAA, FNA, and FSA
    - Ensures solution implemented meets intent of JCIDS analysis

- One place to document entire system lifecycle from inception to deployment
  - Document rationale for decisions and analysis
  - Easily supports changes/updates to the model while maintaining historical information
  - Abstracts the complexities of the warfare system, the capability system, and associated elements such that a team can effectively grasp them

- Appropriate integrated architecture views can be generated
  - Operational views – requirements and functional model
  - System views – physical model
  - Technical views – requirements, physical models
Summary

- Model-driven SE will provide robust lifecycle system model
  - Provides integrated architecture
  - Supports initial capabilities assessment
  - Establishes framework for entire lifecycle: concept refinement, technology development, system development and demonstration, production and deployment, and operations and support phases

- Systems Engineering methodology enhances the JCIDS process
  - Models abstract complexities of modern warfare systems

- Comprehensive models provide for compilation of data needed to assess capabilities and comply with JCIDS

Models bridge the diverse knowledge domains of the warrior and the engineer
Functional Area Analysis (FAA)  
Functional Needs Analysis (FNA)  
Functional Solutions Analysis (FSA)  

Dave Krueger  
Chris Ryder  
Contributions from Lee Kennedy and Bob Finlayson
JCIDS Process*
The Defense Acquisition Management Framework*

- Process entry at Milestones A, B, or C
- Entrance criteria met before entering phase
- Evolutionary Acquisition or Single Step to Full Capability

*DoDI 5000.2, 12 May 2003
The Defense Acquisition Management Framework
Discussion Topics

- Perspective
- Introduction to the TBDA Case
- Functional area analysis (FAA)
- Functional needs analysis (FNA)
- Functional solutions analysis (FSA)
- Post independent analysis (PIA)
- Conclusion
JCIDS Process

DOD Strategic Guidance

Joint Operations Concepts

Joint Operating Concepts
Joint Functional Concepts
Joint Integrating Concepts

Functional Area Analysis

Functional Needs Analysis

JCD

DOTLPF Analysis (Non-materiel Approaches)

Ideas for Materiel Approaches

Analysis of Materiel/Non-materiel Approaches

Alternative N
Alternative 2
Alternative 1

Post Independent Analysis

Integrated Architectures

CPD

CDD

ICD

DCR

* CJCSM 3170.01B
Functional Area Analysis (FAA)

- Produces a list of capabilities across all functional areas necessary to achieve military objectives

- Capabilities
  - Operational tasks
  - Conditions
  - Standards (or measures of effectiveness)

- Input
  - National strategies, JOCs, JFCs, JICs, the Universal Joint Task List (UJTL)
  - Anticipated adversary capabilities
Defining the Problem Space

The Capability Context

- Understanding the Capability Enterprise
  - Environment
  - Enemy forces and systems
- Capability operations
- Capability operators (Warriors)
- Network requirements
- Capability command and control
  - Command authority
- Analysis of legacy Warfare Systems contributing to the Capability Object
- Preliminary Non-Materiel issues
  - DOTMLPF analysis primarily occurs during the FSA
Architecture Meta-Model

class Architecture Model

- Architecture
- Architectural Element
- Dynamic Element
- Physical Element

«DataType» Architecture Model
-  View
- Association
- Requirements Model
- Requirements Traceability Matrix
- Functional Model
- Physical Model
Capability Context
(Repeat from Model Driven SE Section)
FAA Activity Diagram

- National strategies, JOC, JFC, JIC, UJTL, and the anticipated range of broad capabilities that adversaries might employ.
- Collect Relevant Capability Artifacts
- Information Gathering
- Environmental conditions, timeline, threat analysis
- The basis for Mission Level Use Cases and Scenarios.
- These tasks are foundation of "To-Be" Enterprise Model.
- Includes preliminary Capability requirements (operational needs), actors (warriors), tasks (activities) and Capability Context.
- The basis for Mission Level Use Cases and Scenarios.
- These tasks are foundation of "To-Be" Enterprise Model.
- Includes preliminary Capability requirements (operational needs), actors (warriors), tasks (activities) and Capability Context.
- The basis for Mission Level Use Cases and Scenarios.
- These tasks are foundation of "To-Be" Enterprise Model.
- Includes preliminary Capability requirements (operational needs), actors (warriors), tasks (activities) and Capability Context.
- The basis for Mission Level Use Cases and Scenarios.
- These tasks are foundation of "To-Be" Enterprise Model.
- Includes preliminary Capability requirements (operational needs), actors (warriors), tasks (activities) and Capability Context.
- The basis for Mission Level Use Cases and Scenarios.
- These tasks are foundation of "To-Be" Enterprise Model.
- Includes preliminary Capability requirements (operational needs), actors (warriors), tasks (activities) and Capability Context.
- The basis for Mission Level Use Cases and Scenarios.
- These tasks are foundation of "To-Be" Enterprise Model.
- Includes preliminary Capability requirements (operational needs), actors (warriors), tasks (activities) and Capability Context.
- The basis for Mission Level Use Cases and Scenarios.
- These tasks are foundation of "To-Be" Enterprise Model.
- Includes preliminary Capability requirements (operational needs), actors (warriors), tasks (activities) and Capability Context.
- The basis for Mission Level Use Cases and Scenarios.
- These tasks are foundation of "To-Be" Enterprise Model.
- Includes preliminary Capability requirements (operational needs), actors (warriors), tasks (activities) and Capability Context.
- The basis for Mission Level Use Cases and Scenarios.
- These tasks are foundation of "To-Be" Enterprise Model.
- Includes preliminary Capability requirements (operational needs), actors (warriors), tasks (activities) and Capability Context.
- The basis for Mission Level Use Cases and Scenarios.
- These tasks are foundation of "To-Be" Enterprise Model.
- Includes preliminary Capability requirements (operational needs), actors (warriors), tasks (activities) and Capability Context.
- The basis for Mission Level Use Cases and Scenarios.
- These tasks are foundation of "To-Be" Enterprise Model.
- Includes preliminary Capability requirements (operational needs), actors (warriors), tasks (activities) and Capability Context.
- The basis for Mission Level Use Cases and Scenarios.
- These tasks are foundation of "To-Be" Enterprise Model.
- Includes preliminary Capability requirements (operational needs), actors (warriors), tasks (activities) and Capability Context.
- The basis for Mission Level Use Cases and Scenarios.
- These tasks are foundation of "To-Be" Enterprise Model.
- Includes preliminary Capability requirements (operational needs), actors (warriors), tasks (activities) and Capability Context.
- The basis for Mission Level Use Cases and Scenarios.
- These tasks are foundation of "To-Be" Enterprise Model.
- Includes preliminary Capability requirements (operational needs), actors (warriors), tasks (activities) and Capability Context.
- The basis for Mission Level Use Cases and Scenarios.
- These tasks are foundation of "To-Be" Enterprise Model.
- Includes preliminary Capability requirements (operational needs), actors (warriors), tasks (activities) and Capability Context.
- The basis for Mission Level Use Cases and Scenarios.
- These tasks are foundation of "To-Be" Enterprise Model.
- Includes preliminary Capability requirements (operational needs), actors (warriors), tasks (activities) and Capability Context.
- The basis for Mission Level Use Cases and Scenarios.
- These tasks are foundation of "To-Be" Enterprise Model.
- Includes preliminary Capability requirements (operational needs), actors (warriors), tasks (activities) and Capability Context.
- The basis for Mission Level Use Cases and Scenarios.
- These tasks are foundation of "To-Be" Enterprise Model.
FAA Approach

Joint Operations Concepts (JOpsC)
- Fully Integrated
- Expeditionary
- Networked
- Decentralized
- Adaptable
- Decision superiority
- Lethality

Joint Functional Concepts (JFC)
- Force application
- Force protection
- Focused logistics
- Battlespace awareness
- Command and control
- Force management
- Net centric
- Joint training

Joint Integrating Concepts (JIC)
- Forceable Entry Ops
- Undersea Superiority
- Global Strike Ops
- Sea-Basing Ops
- Air & Missile Defense
- JC2
- Joint Logistics

Joint Operating Concepts (JOC)
- Homeland Security
- Stability Operations
- Strategic Deterrence
- Major Combat Operations

Universal Joint Task List (UJTL)
- Strategic National (SN)
- Strategic Theater (ST)
- Operational (OP)
- Tactical (TA)

Universal Navy Task List (UNTL)
- UJTL (Strategic & Operational)
- Navy Tactical Task List (NTTL)

Required Capabilities

Applicable, Existing FAAs
- Battlespace Awareness FAA

Program Specific Documentation

Other Input
- Operational personnel
- Customers
Joint Operations Concepts (JOpsC)

- An overarching description of how the future Joint Force will operate across the entire range of military operations

- Attributes
  - Fully Integrated
  - Expeditionary
  - Networked
  - Decentralized
  - Adaptable
  - Decision superiority
  - Lethality

Too general for specific FAA development
Joint Integrating Concepts (JICs)

- Description of how a Joint Force Commander will integrate capabilities to generate effects and achieve an objective
  - Forceable Entry Ops
  - Undersea Superiority
  - Global Strike Ops
  - Sea-Basing Ops
  - Air & Missile Defense
  - JC2
  - Joint Logistics

- Includes an illustrative CONOPS for a specific scenario and a set of distinguishing principles applicable to a range of scenarios
Joint Operating Concepts (JOCs)

- Operational-level description of how the Joint Force Commander will operate and a foundation for defining military capabilities

- Operational context for JFC and JIC development
  - Homeland Security (HLS)
  - Stability Operations (SO)
  - Strategic Deterrence (SD)
  - Major Combat Operations (MCO)
Joint Functional Concepts (JFCs)

- Describes how the joint force will perform military functions across the range of military operations
- Functional areas
  - Force application
  - Force protection
  - Focused logistics
  - Battlespace awareness
  - Command and control
  - Force management
  - Net centric
  - Joint training

Functional Capability Board (FCB) for each functional area
Functional Capability Boards (FCB)

- Responsible for organization, analysis, and prioritization of capability needs proposals within their functional areas

- Provide oversight and assessment throughout JCIDS process
  - Reduce redundant analyses
  - Ensure consistency in capability definitions
  - Ensure approaches consider a broad range of possibilities

- Provides context briefing to JROC
  - Where capability proposal fits within functional area

- Make recommendations on validation and approval

Identify appropriate FCB and involve them in the analyses!
Battlespace Awareness JFC Capabilities

- **Operational**
  - Command and control of BA assets
  - Execute collection
  - Exploitation and analysis
  - M&S, forecast
  - Manage knowledge

- **Enabling**
  - Integrate BA network
  - Infuse emergent technology
  - Recruit, retain, train

Several capabilities IDed for each capability category, e.g.,
- Surveillance
- Cross cue
- Employ human resources
- Employ open source resources
- Measure & monitor environmental conditions

Battlespace Awareness FAA (Draft)

- Defines Battlespace Awareness capabilities for each task and sub-task in each JOC
  - Homeland Security (HLS)
  - Stability Operations
  - Strategic Deterrence
  - Major Combat Operations (MCO)

1. Interdiction
   a. Kill 1st echelon forces
   b. Divert/delay follow-on forces
2. Ground operations
3. Air defense
4. Missile defense
5. JSEAD
6. Strike
7. Sea Strike Operations
8. Sea Shield Operations
9. Sea Basing Operations
10. Info Operations
11. Battlespace Awareness
12. Intent/I&W
13. I&W Specific Threat

Tasks and sub-tasks for other JOCs not shown
UJTL and UNTL

- **UJTL:** “The Universal Joint Task List (UJTL), when augmented with the Service task lists, is a comprehensive integrated **menu** of **functional tasks, conditions, measures, and criteria** supporting all levels of the Department of Defense in executing the National Military Strategy.”

- **UNTL:** “The UNTL tasks make up a comprehensive hierarchical structure. The UNTL task list is designed to be **comprehensive** while being **mutually exclusive**. When reviewing the levels of the hierarchy, the subordinate tasks will, in total, comprehensively, and without redundancy, define all activities involved in the next higher-level task.”
Universal Joint Task List (UJTL)*
Levels of War

- (SN) Strategic level - National military tasks
  - Accomplish objectives of national military strategy
- (ST) Strategic level - Theater tasks
  - Accomplish objectives of the theater and campaign strategy
- (OP) Operational level tasks
  - Accomplish objectives of subordinate campaigns and major operations
- (TA) Tactical level tasks - include joint/interoperability tactical tasks and the applicable Service tasks
  - Accomplish objectives of battles and engagements

* CJCSM 3500.04C, “Universal Joint Task List (UJTL),” 1 July 2002
**Example Hierarchy**

To obtain, by various detection methods, information about the activities of an enemy or potential enemy or tactical area of operations. This task uses surveillance to systematically observe the area of operations by visual, aural, electronic, photographic, or other means. This includes development and execution of search plans.

<table>
<thead>
<tr>
<th>NTA</th>
<th>Navy Tactical</th>
</tr>
</thead>
<tbody>
<tr>
<td>NTA 1</td>
<td>Deploy Forces/Conduct Maneuver</td>
</tr>
<tr>
<td>NTA 2</td>
<td>Develop Intelligence</td>
</tr>
<tr>
<td>NTA 2.1</td>
<td>Plan and Direct Intelligence Operations</td>
</tr>
<tr>
<td>NTA 2.2</td>
<td>Collect Data and Intelligence</td>
</tr>
<tr>
<td>NTA 2.2.1</td>
<td>Collect Target Information</td>
</tr>
<tr>
<td>NTA 2.2.2</td>
<td>Collect Tactical Intelligence on Situation</td>
</tr>
<tr>
<td>NTA 2.2.3</td>
<td>Perform Tactical Reconnaissance and Surveillance</td>
</tr>
<tr>
<td>NTA 2.2.4</td>
<td>Assess Tactical Environment</td>
</tr>
<tr>
<td>NTA 2.3</td>
<td>Process and Exploit Collected Info/Intelligence</td>
</tr>
<tr>
<td>NTA 2.3.1</td>
<td>Conduct Technical Processing and Exploitation</td>
</tr>
<tr>
<td>NTA 2.3.2</td>
<td>Correlate Information</td>
</tr>
<tr>
<td>NTA 2.4</td>
<td>Produce Intelligence</td>
</tr>
<tr>
<td>NTA 2.5</td>
<td>Disseminate and Integrate Intelligence</td>
</tr>
<tr>
<td>NTA 3</td>
<td>Employ Firepower</td>
</tr>
<tr>
<td>NTA 4</td>
<td>Perform Logistics and Combat Service Support</td>
</tr>
<tr>
<td>NTA 5</td>
<td>Exercise Command and Control</td>
</tr>
<tr>
<td>NTA 6</td>
<td>Protect the Force</td>
</tr>
</tbody>
</table>

To associate and combine data on a single subject to improve the reliability or credibility of the information. This task includes collating information (identifying and grouping related items of information for critical comparison).
Identify Tasks for FNA

- Select Tasks from UJTL
- Select Tasks from FCB Portfolio

- Define Standards and Conditions for Each Task
  - Capability Task List
    - Task 1..N
  - Task Measures List
    - Measure 1..N

- «DataType»
  - UJTL
    - Strategic: National: SN-1..N
    - Strategic: Theater: ST-1..N
    - Operational: OP-1..N
    - Tactical: TA-1..N
Identify Tasks

Joint Operations Concepts (JOpsC)
- Fully Integrated
- Expeditionary
- Networked
- Decentralized
- Adaptable
- Decision superiority
- Lethality

Joint Functional Concepts (JFC)
- Force application
- Force protection
- Focused logistics

- Battlespace awareness
- Command and control
- Force management
- Net centric
- Joint training

Joint Operating Concepts (JOC)
- Homeland Security
- Stability Operations
- Strategic Deterrence
- Major Combat Operations

Joint Integrating Concepts (JIC)
- Force application
- Force protection
- Focused logistics

- Global Strike Ops
- Sea-Basing Ops
- Air & Missile Defense
- JC2
- Joint Logistics

Universal Joint Task List (UJTL)
- Strategic National (SN)
- Strategic Theater (ST)
- Operational (OP)
- Tactical (TA)

Universal Navy Task List (UNTL)
- UJTL
- Navy Tactical Task List (NTTL)

Applicable, Existing FAAs
- Battlespace Awareness FAA

Program Specific Documentation

Other Input
- Operational personnel
- Customers

Required Capabilities

Identify tasks
## Select Tasks from UJTL & UNTL

<table>
<thead>
<tr>
<th>NTA</th>
<th>Navy Tactical</th>
<th>Y/P</th>
</tr>
</thead>
<tbody>
<tr>
<td>NTA 1</td>
<td>Deploy Forces/Conduct Maneuver</td>
<td>P</td>
</tr>
<tr>
<td>NTA 2</td>
<td>Develop Intelligence</td>
<td>Y</td>
</tr>
<tr>
<td>NTA 2.1</td>
<td>Plan and Direct Intelligence Operations</td>
<td>N</td>
</tr>
<tr>
<td>NTA 2.2</td>
<td>Collect Data and Intelligence</td>
<td>Y</td>
</tr>
<tr>
<td>NTA 2.2.1</td>
<td>Collect Target Information</td>
<td>Y</td>
</tr>
<tr>
<td>NTA 2.2.2</td>
<td>Collect Tactical Intelligence on Situation</td>
<td>Y</td>
</tr>
<tr>
<td>NTA 2.2.3</td>
<td>Perform Tactical Reconnaissance and Surveillance</td>
<td>Y</td>
</tr>
<tr>
<td>NTA 2.2.4</td>
<td>Assess Tactical Environment</td>
<td>Y</td>
</tr>
<tr>
<td>NTA 2.3</td>
<td>Process and Exploit Collected Info/Intelligence</td>
<td>Y</td>
</tr>
<tr>
<td>NTA 2.3.1</td>
<td>Conduct Technical Processing and Exploitation</td>
<td>Y</td>
</tr>
<tr>
<td>NTA 2.3.2</td>
<td>Correlate Information</td>
<td>Y</td>
</tr>
<tr>
<td>NTA 2.4</td>
<td>Produce Intelligence</td>
<td>Y</td>
</tr>
<tr>
<td>NTA 2.5</td>
<td>Disseminate and Integrate Intelligence</td>
<td>Y</td>
</tr>
<tr>
<td>NTA 3</td>
<td>Employ Firepower</td>
<td>P</td>
</tr>
<tr>
<td>NTA 4</td>
<td>Perform Logistics and Combat Service Support</td>
<td>N</td>
</tr>
<tr>
<td>NTA 5</td>
<td>Exercise Command and Control</td>
<td>P</td>
</tr>
<tr>
<td>NTA 6</td>
<td>Protect the Force</td>
<td>P</td>
</tr>
</tbody>
</table>
## Select Tasks from FCB Portfolios

<table>
<thead>
<tr>
<th>Battlespace Awareness Tasks</th>
<th>Applicable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command and Control of BA Assets</td>
<td><img src="https://example.com/applicable.png" alt="Applicable" /></td>
</tr>
<tr>
<td>Task, dynamically re-task and monitor assets</td>
<td><img src="https://example.com/applicable.png" alt="Applicable" /></td>
</tr>
<tr>
<td>Execute Collection</td>
<td><img src="https://example.com/applicable.png" alt="Applicable" /></td>
</tr>
<tr>
<td>Cross cue</td>
<td><img src="https://example.com/applicable.png" alt="Applicable" /></td>
</tr>
<tr>
<td>Employ human resources</td>
<td><img src="https://example.com/applicable.png" alt="Applicable" /></td>
</tr>
<tr>
<td>Employ open source resources</td>
<td><img src="https://example.com/applicable.png" alt="Applicable" /></td>
</tr>
<tr>
<td>Measure and monitor environmental conditions</td>
<td><img src="https://example.com/applicable.png" alt="Applicable" /></td>
</tr>
<tr>
<td>Exploit and Analyze</td>
<td><img src="https://example.com/applicable.png" alt="Applicable" /></td>
</tr>
<tr>
<td>Recognize targets</td>
<td><img src="https://example.com/applicable.png" alt="Applicable" /></td>
</tr>
<tr>
<td>Distribute processing</td>
<td><img src="https://example.com/applicable.png" alt="Applicable" /></td>
</tr>
<tr>
<td>Information fusion</td>
<td><img src="https://example.com/applicable.png" alt="Applicable" /></td>
</tr>
<tr>
<td>Enable analyst collaboration</td>
<td><img src="https://example.com/applicable.png" alt="Applicable" /></td>
</tr>
<tr>
<td>Defeat denial and deception</td>
<td><img src="https://example.com/applicable.png" alt="Applicable" /></td>
</tr>
<tr>
<td>Model, Simulate, Forecast/Predict</td>
<td><img src="https://example.com/applicable.png" alt="Applicable" /></td>
</tr>
<tr>
<td>Predictive analysis</td>
<td><img src="https://example.com/applicable.png" alt="Applicable" /></td>
</tr>
<tr>
<td>Integrate adversary and friendly information</td>
<td><img src="https://example.com/applicable.png" alt="Applicable" /></td>
</tr>
<tr>
<td>Manage Knowledge</td>
<td><img src="https://example.com/applicable.png" alt="Applicable" /></td>
</tr>
<tr>
<td>Smart pull/push information</td>
<td><img src="https://example.com/applicable.png" alt="Applicable" /></td>
</tr>
<tr>
<td>Share plan visibility</td>
<td><img src="https://example.com/applicable.png" alt="Applicable" /></td>
</tr>
<tr>
<td>Content management</td>
<td><img src="https://example.com/applicable.png" alt="Applicable" /></td>
</tr>
</tbody>
</table>
Define Conditions and Standards for Each Task

- **Conditions** - a variable of the environment that affects performance of a task
  - Physical: land, sea, air, space
  - Military: mission; forces; C3; intelligence; deployment, movement, and maneuver; firepower; protection; sustainment; threat; conflict
  - Civil: political policies, culture, economy,

- **Standard** - the minimum proficiency required in the performance of a task
  - **Measure** - Quantitative or qualitative basis for describing the quality of task performance
  - **Criterion** - A critical, threshold, or specified value of a measure

- **Sources**
  - UJTL/UNTL
  - Design Reference Mission (DRM)
  - Subject Matter Experts (SMEs)
Example Conditions From UNTL

- **C 1.0 PHYSICAL ENVIRONMENT**
  - **C 1.1 LAND**
    - **C 1.1.1 Terrain**
      - **C 1.1.1.1 Terrain Relief**
      - **C 1.1.1.2 Terrain Elevation**
      - **C 1.1.1.3 Terrain Slope**
      - **C 1.1.1.4 Terrain Firmness**
      - **C 1.1.1.5 Terrain Traction**
    - **C 1.1.1.6 Vegetation**
      - Plants, trees, and shrubs.
      - *Descriptors:* Jungle (rainforest, canopied); Dense (forested); Light (meadow, plain); Sparse (alpine, semi-desert); Negligible (arctic, desert).
    - **C 1.1.1.7 Terrain Relief features**
    - **C 1.1.2 Geological Features**
    - **C 1.1.3 Man-Made Terrain Features**
    - Etc.
NTA 2.2.3 Perform Tactical Reconnaissance and Surveillance

To obtain, by various detection methods, information about the activities of an enemy or potential enemy or tactical area of operations. This task uses surveillance to systematically observe the area of operations by visual, aural, electronic, photographic, or other means. This includes development and execution of search plans.

<table>
<thead>
<tr>
<th>Units</th>
<th>Measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Days</td>
<td>From receipt of tasking, until reconnaissance/surveillance assets in place</td>
</tr>
<tr>
<td>Percent</td>
<td>Of collection requirements fulfilled by reconnaissance/surveillance assets</td>
</tr>
<tr>
<td>Percent</td>
<td>Of time able to respond to collection requirements</td>
</tr>
<tr>
<td>Hours</td>
<td>To respond to emergent tasking</td>
</tr>
<tr>
<td>Percent</td>
<td>Operational availability of tactical aircraft reconnaissance systems</td>
</tr>
<tr>
<td>Time</td>
<td>To exploit single tasked image collected after aircraft on deck</td>
</tr>
</tbody>
</table>
**FAA Results**

Description of the operational/tactical situation including the appropriate conditions: 3 days from receipt of tasking, until reconnaissance/surveillance assets in place.

<table>
<thead>
<tr>
<th>Mission Type 1</th>
<th>Task 1.1</th>
<th>Standard 1.1.1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Task 1.2</td>
<td>Standard 1.2</td>
</tr>
<tr>
<td></td>
<td>Task 1.3</td>
<td>Standard 1.3</td>
</tr>
<tr>
<td></td>
<td>Task 1.4</td>
<td>Standard 1.4</td>
</tr>
<tr>
<td></td>
<td>Task 1.5</td>
<td>Standard 1.5.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Standard 1.5.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Standard 1.5.3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Mission Type 2</th>
<th>Missions Type 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mission Type 4</td>
<td></td>
</tr>
</tbody>
</table>

Description of the task, e.g., “Perform tactical reconnaissance and surveillance”:

6 hours to exploit single tasked image collected after aircraft on deck.
FAA Artifacts

- First iteration of Architecture Model
  - Capability Requirements Model
  - Capability Context Diagram
    - Block Definition Diagram
  - Identification of actors within the Capability Enterprise
    - Block Definition Diagram
  - Capability tasks depicted in Use Case Diagram
    - And Activity Diagram as appropriate
FAA Artifacts

- First Iteration of Architecture Model
  - Initial information exchanges and data elements
    - Sequence Diagram
  - Tasks can be captured as SysML Blocks that include task standards
    - Task attributes (measures)
    - Results of Value (Post-Conditions that determine success/failure)

- Capability Task List
  - Capability tasks trace to model elements, including capability requirements
TBDA FAA

- Define the operational enterprise for the TBDA that will:
  - Establish measurable capability needs for targeting and surveillance
  - Specify the pertinent operations for targeting and surveillance that will capture the correct capability tasks
  - Comprehend the “Capability Enablers”, those “things” that provide pieces of the overall capability
    - Including initial review of legacy systems
    - As well as non-materiel contributors
  - Who are the beneficiaries of the capability
  - What are the potential data elements, information exchanges and interfaces
  - Verify that the capability is required and that it is captured correctly
TBDA Context

- TBDA Context
  - TBDA Warrior
    - Operates
  - Planning Element
    - Plans Missions
  - Command Authority
    - Directs
  - TBDA Logistics & Support Operatives
    - Supports
    - Operated by
  - «System» Global Information Grid
    - Supports
  - «System» Supported Strike System
    - Supported by
  - «System» TBDA Target
    - Sensed by
TBDA Context

- At this point in the analysis, the TBDA:
  - Is under the direction of some Command Authority
  - Senses a generic class of targets
  - Interfaces with the Global Information Grid
  - Operated by Warriors
  - Supported and maintained by Warriors

- It is highly probable that the Capability Context will be modified throughout the FAA (and follow-on analysis phases)
TBDA Capability Elements  
(The FAA Requirements Model)

- Four basic needs categories
  - Networking/ Data Link
    - TBDA Information must “get somewhere”
  - Sensors
    - Some “thing” must capture TBDA information
  - Vehicle
    - The capability must be deployed within a defined battespace
  - Supportability and Logistics
    - “Professionals think Logistics”

- Requirements can be captured using data bases and graphical tools
  - Requirements Traceability Matrix (RTM)
  - SysML Requirements Diagram
  - These artifacts must be “tightly coupled”
TBDA FAA Requirements Diagram

SysML diagrams supported by robust database
TBDA Fundamental Operations

SysML Use Case Diagram Captures Basic Functionality Performed by Actors
Fundamental Operations Function Tree

class TBDA Functional Hierarchy

«Use Case» Conduct TBDA Mission

- «Use Case» Develop the TBDA Surveillance Mission
- «Use Case» Transport the TBDA
- «Use Case» Collect, Process and Assess Surveillance Information
- «Use Case» Disseminate the TBDA Information
- «Use Case» Support and Sustain the TBDA
TBDA Functional Decomposition
(Develop the TBDA Mission)

- Collect Relevant Pre-flight Data
- Conduct Preliminary Mission Analysis
- Develop Communications Network
- Develop Sensor Plan
- Develop Vehicle Operations Plan
- Develop Integrated Pre-Mission Plan
- Download Mission Plan to TBDA Vehicle
- «DataType» Operational Information
- «DataType» TBDA Mission Tasking
- «DataType» TBDA Mission Tasking
- Mission Plan

Tasks:
- Act Mission Development
- Comm Plan
- Sensor Plan
- Vehicle Plan
TBDA Functional Analysis
(Initial Data Exchanges)

Command Authority

Mission Development Team

TBDA Supporting Elements

TBDA Warrior

Mission Tasking

Request Operational Information

Operational Information

Mission Plan Ready for Download to TBDA
SysML Diagrams Supporting Functional Analysis

- **Activity Diagram**
  - Depicts functional elements as activities
    - Activities create Data Elements that are consumed in subsequent activities or use cases
    - Team can assess initial data requirements
    - Data Element is most often the Result of Value (ROV)
  - Modified to show “who” is performing the activities via “swim lanes”

- **Sequence Diagram (aka Interaction Diagram)**
  - Depicts sequence of information exchanges
    - Sending and receiving nodes
  - Analysis team should be able to get an initial understanding of interface requirements

Activities and Use Cases Trace to UNTL Tasks
TBDA FAA Logical Elements

Class TBDA Capability

- Legacy System
  - Legacy Attributes (1..N)
  - Legacy Operations (1..N)

Applies 0..*

- Capability Enabler
  - TBDA Support Element
  - TBDA Controls
  - TBDA Sensor
  - TBDA Vehicle
  - TBDA Communications

Capability Enabler Non-Material
Even during the FAA, some “Capability Enablers” can be logically deduced
- As analysis progresses, the attributes for these Enablers will be defined as well as functionality assigned to those elements

Acknowledged that there are some non-materiel contributors

Initial review of legacy systems
- Further studied during FNA
**TBDA FAA Architectural Model**

- Captured the operational need through a SysML Requirements Diagram
  - Along with RTM
  - SE Method: Requirements Analysis
- Identified basic functionality
  - Will contribute to UJTL task assessment
  - Initially capture potential data elements and interfaces
  - Help generate Capability Task List for the FNA
  - SE Method: Functional Definition
- Identified Capability Enablers
  - Along with “first look” contributions by Legacy Systems and non-materiel elements
  - SE Method: Physical Definition
- Each of the basic functions and Capability Enablers must trace to the capability requirements listed in the Statement of Operational Need
  - SE Method: Design Validation
JCIDS Process

- DOD Strategic Guidance
- Joint Operations Concepts
- Joint Operating Concepts
- Joint Functional Concepts
- Joint Integrating Concepts

Functional Area Analysis

- Functional Needs Analysis
- DOTLPF Analysis (Non-materiel Approaches)
- Ideas for Materiel Approaches
- Analysis of Materiel/Non-materiel Approaches

Integrated Architectures

- Alternative N
- Alternative 2
- Alternative 1

Post Independent Analysis

- CPD
- CDD
- ICD
- DCR

Functional Solution Analysis

*CJCSM 3170.01B*
Functional Needs Analysis (FNA)

- Assess current and programmed warfighting systems
  - Can they deliver the capabilities identified in the FAA
  - Uses conditions from FAA
  - Uses standards from the FAA as the “measuring stick”

- Output
  - List of capability gaps or shortfalls
    - Relative priority
    - Timeframe for required solutions
  - Identify redundancies in capabilities that reflect inefficiencies.

Determines gaps in planned capabilities
FNA Activities

- **Activity/Task Mapping**
  - Functional decomposition including measurable results of value
  - Assignment of functions to logical elements
  - Refinement of capability measures
    - Forwarded to JROC for approval

- **Resource allocation**
  - Contribution of legacy Warfare System capabilities

- **Trades Analyses**
  - Cost, schedule and performance constraints
  - Task alternatives and weighting

- **Interoperability assessment**
  - Refinement of information exchanges and date elements
FNA Functional Analysis

- Expand functional analysis from FAA
- Identify data elements/ data attributes
- Consider interface elements such as communications links
- Quantify measurable use case results of value
- Functional contributions of legacy systems and subsystems
Functional Analysis with Use Cases

- Use Case
  - Sequence of events that returns a measurable Result of Value (Booch, Rumbaugh, Jacobson)
  - Captures
    - Actors (Warriors performing the activities)
      - Roles, not specific individuals or commands
    - Activities (operations)
    - Data objects
      - Created and consumed by the activities
      - Information elements that are exchanged between the Operational Nodes
    - Any other relevant references such as UJTL tasks

Functional Analysis is basic to the Systems Engineering Method
FNA Activity Diagram

act Perform FNA

- Assess Capability Gaps, Overlaps and operational problem(s)
- Analyze potential new Functional Areas for problem or solution
- Formulate key attributes for capability development of Measures of Effectiveness
- Identify Key Architectural Elements affected by new capability
- JROC Approved
- From FAA: Architecture Model
- Resource List
  - Financial Resources: Money
  - Warfare Systems: Support Systems
- Capability Task List
  - Mission 1..N
  - Task 1..N
  - Standard
  - Measure 1..N
  - Traceability to National Strategy
- Capability Gap List
  - Capability Gap 1..N
- Capability Attribute
  - Weight:
  - Risk:
- Architecture Model
- Use Case ROV
- Use Case
- Functional Model
- Capability Task Weight
- Capability Gap List
  - Capability Gap 1..N
- Capability Element
  - Weight:
  - Risk:
## Example FNA Methodology

1. Assign weights based on relative importance

2. Select alternatives (screen based on FAA)

3. Evaluate performance of each alternative for each mission, task, and standard

4. Scores less than one indicate capability gaps

<table>
<thead>
<tr>
<th>FAA</th>
<th>Weight</th>
<th>Alternative A Score</th>
<th>Alternative B Score</th>
<th>Alternative C Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mission Type 1</td>
<td>0.30</td>
<td>0.79</td>
<td>0.79</td>
<td>0.69</td>
</tr>
<tr>
<td>Task 1.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Standard 1.1.1</td>
<td>0.10</td>
<td>0.90</td>
<td>0.80</td>
<td>0.85</td>
</tr>
<tr>
<td>Standard 1.1.2</td>
<td>0.05</td>
<td>0.85</td>
<td>0.90</td>
<td>0.60</td>
</tr>
<tr>
<td>Task 1.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Standard 1.2</td>
<td>0.15</td>
<td>0.90</td>
<td>0.70</td>
<td>0.70</td>
</tr>
<tr>
<td>Task 1.3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Standard 1.3</td>
<td>0.30</td>
<td>0.80</td>
<td>0.80</td>
<td>0.80</td>
</tr>
<tr>
<td>Task 1.4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Standard 1.4</td>
<td>0.10</td>
<td>0.50</td>
<td>0.50</td>
<td>0.40</td>
</tr>
<tr>
<td>Task 1.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Standard 1.5.1</td>
<td>0.05</td>
<td>0.70</td>
<td>0.70</td>
<td>0.50</td>
</tr>
<tr>
<td>Standard 1.5.2</td>
<td>0.20</td>
<td>0.75</td>
<td>0.95</td>
<td>0.70</td>
</tr>
<tr>
<td>Standard 1.5.3</td>
<td>0.05</td>
<td>0.95</td>
<td>0.90</td>
<td>0.50</td>
</tr>
<tr>
<td>Mission Type 2</td>
<td>0.30</td>
<td>0.90</td>
<td>0.70</td>
<td>0.80</td>
</tr>
<tr>
<td>Mission Type 3</td>
<td>0.30</td>
<td>0.80</td>
<td>0.90</td>
<td>0.70</td>
</tr>
<tr>
<td>Mission Type 4</td>
<td>0.10</td>
<td>0.70</td>
<td>0.90</td>
<td>0.60</td>
</tr>
<tr>
<td><strong>Composite Score</strong></td>
<td></td>
<td><strong>0.82</strong></td>
<td><strong>0.81</strong></td>
<td><strong>0.72</strong></td>
</tr>
</tbody>
</table>
FNA Artifacts

- Next iteration of Architecture Model including
  - Definition of Capability Elements
    - Model as Blocks including attributes, parameters and constraints
  - Definition of legacy systems that contribute to the capability
  - Capturing functional tasks as use cases with ROVs
  - Assignment of functional tasks to Capability (logical) Elements
  - Use cases assigned to Capability Elements as Block operations

- Capability measures forwarded to JROC for approval
- Requirements Traceability Matrix
To what extent does the Legacy attributes and operations satisfy the capability need, including UJTL tasks?
Existing subsystems/components with suitable performance measures may provide some capability requirements, but will need to be integrated into an overall solution.
Key Architectural Elements

- In the FNA, architectural elements are still abstractions (i.e. capability enablers) of real systems
- Architectures include behaviors, relationships AND rules for “rules governing their design over time” (DoDAF)
  - FNA is the time to consider the “possible” with regard to applications for the capability enablers
    - Including the standards that govern the application
- Example, TBDA Communications
  - Tactical application places limits on size, weight, range of operations
  - Logical conclusion (after analysis): Comm Applications limited to Link-16, UHF/ VMF and CDL
    - Standardized interfaces exist for those applications
    - Modeled using SysML Internal Block Diagram
TBDA FNA Communications Architecture

composite structure Comm_Interfaces

«System»
Global Information Grid

Link-16/ Mil-Std-6016
UHF/ VMF
UHF/ VMF/ Mil-Std-6017
Common Data Link

Port: Interface defined by Standard

«Capability Enabler»
TBDA Communications

Link-16/ Mil-Std-6016
UHF/ VMF
UHF/ VMF/ Mil-Std-6017
Common Data Link
TBDA FNA Model

- Define operational functionality and assign that functionality to the logical elements, i.e. Capability Enablers
- Define capability attributes with suitable measures of effectiveness
  - Assign measures to key architectural elements
- Do these elements satisfy the identified capability gaps?
  - If not, refine
JCIDS Process

- DOD Strategic Guidance
  - Joint Operations Concepts
    - Joint Operating Concepts
      - Joint Functional Concepts
      - Joint Integrating Concepts
    - Functional Area Analysis
      - Functional Needs Analysis
        - JCD
      - DOTLPF Analysis (Non-materiel Approaches)
      - Ideas for Materiel Approaches
      - Analysis of Materiel/Non-materiel Approaches
        - Alternative N
          - Alternative 2
          - Alternative 1
    - Integrated Architectures
      - CPD
      - CDD
      - ICD
      - DCR
    - Post Independent Analysis
  - Functional Solution Analysis

*CJCSM 3170.01B*
Functional Solution Analysis (FSA)

- Operational assessment of all approaches to solving the capability gaps identified in the FNA
  - Non-materiel solutions
  - Materiel solutions (in priority order)
    - Product improvements to existing materiel or facilities
    - Adoption of interagency or foreign materiel solutions
    - Initiation of new materiel programs

- Basis for ICD

Transition from Problem Space to Solution Space
FSA Activities

- Define the Solution Space
  - Trace possible solutions to satisfactory “solve the problem”
- Conduct the DOTMLPF analysis
  - Model DOTMLPF Elements as Blocks that include attributes and operations
- Refine Use Cases and appropriate ROVs after DOTMLPF factored into Solution Space
- Analyze potential material solutions, i.e. Warfare Systems
  - Model Warfare Systems as logical elements and assign use cases – Assignment of functionality to physical elements
FSA Activities

- Analysis of Material Alternatives (AMA)
  - Analyze Capability Gap and range of military operations
  - Assess operational risk and DOTMLPF implications
  - Assess material impact to functional areas
- Program Independent Analysis (PIA)
  - Ensure the list of approaches with the potential to deliver the capability identified in the FAA and FNA is complete
The AMA is significant enough to have a separate Use Case to denote the series of activities.

Updated list of Material and non-material approaches from AMA.
FSA: Non-Materiel Solutions

- Can FNA capability gaps be mitigated via a non-materiel solution (DOT_LPF)
  - Doctrine
  - Organization
  - Training
  - Leadership/education
  - Personnel
  - Facilities

Generally a qualitative assessment
DOTMLPF as a Logical Element

class DOTMLPF

«Capability»
Capability Object

«DataType»
DOTMLPF Element

Contributes to

«DataType»
Doctrine

«DataType»
Organization

«DataType»
Training

«DataType»
Leadership

«DataType»
Personnel

«DataType»
Facility
Doctrine “Block”

```
class Doctrine

  «DataType»
  Doctrine

  Attribute
    Force
    Data
    Policy
    Material

  Operation
    Employment
    Sustainment
    Comms
    Tactics

  Intel
```
FSA: Materiel Solutions
Analysis of Materiel Approaches (AMA)

- Assess potential materiel solutions to FNA capability gaps
  - Performance
  - Cost
  - Risk

- Some similarity to the Analysis of Alternatives (AoA)
  - Less rigorous
  - Less specific
### Notional FSA Results

<table>
<thead>
<tr>
<th>Approach</th>
<th>Effectiveness</th>
<th>Cost</th>
<th>Risk</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td><img src="image1" alt="yellow" /></td>
<td>$</td>
<td><img src="image2" alt="green" /></td>
</tr>
<tr>
<td>B</td>
<td><img src="image1" alt="yellow" /></td>
<td>$$$</td>
<td><img src="image2" alt="yellow" /></td>
</tr>
<tr>
<td>C</td>
<td><img src="image3" alt="red" /></td>
<td>$</td>
<td><img src="image2" alt="yellow" /></td>
</tr>
<tr>
<td>D</td>
<td><img src="image1" alt="green" /></td>
<td>$$$$</td>
<td><img src="image3" alt="red" /></td>
</tr>
</tbody>
</table>

**Potential non-materiel and material approaches**
FSA Artifacts

- Final Architecture Model
  - “To-Be” model
  - Material elements that perform functional operations that provide capability
    - Measurable Results of Value
- JCIDS Document
  - ICD for Pre-Milestone A
  - CDD and CPD for every capability upgrade
- Concept of Operations (CONOPS)
The “CONOPS” is not a required JCIDS artifact, however
- CONOPS document is a critical interface between the operational and the engineering communities
- Provides potential developers the framework on how the capability will be applied in the operational environment

- Services may require some form of CONOPS
  - USAF Enabling Concept
FSA TBDA “To-Be” System

 TBDA

- Develop the TBDA Surveillance Mission()
- Transport the TBDA()
- Collect, Process and Assess Surveillance Information()
- Disseminate the TBDA Information()
- Support and Sustain the TBDA()

 TBDA Air Vehicle

- Time of Flight: > Three Hours
- Range for Coverage: > 300 KM
- Range of Coverage for Comm: > 400 KM
- Area of Coverage: > 62,500 KM
- Execute Vehicle Transport Functions()

 TBDA Ground Control Station

- Develop the TBDA Surveillance Mission()
- Control TBDA Air Vehicle()
- Process TBDA Information()

 TBDA Sensor System

- Field of View (Staring): = + 30 Degrees
- Sensor Field of View: = + 60 Degrees
- Sensor Range Minimum: = 5KM
- Sensor Range Maximum: = 30 KM
- Minimum RCS: = 2 Square Meters
- Range of Target Speeds: = 5 to 75 MPH
- Capture Surveillance Information()

 TBDA Comm System

- Data Link Effective Range: = 400 KM
- Bandwidth/ Wave Form: = Link-16/ UHF-VMF
- Data Transfer Rate:

+ Disseminate TBDA Information()
+ Interface with GIG()

 TBDA Sustainment Element

- Launch and Landing Range: int = <= 100 Meters
- Set Up Time: <= 30 Minutes
- Mean Maintenance Time: <= 2 hours when ...
- MTBMA: = > 48 Flight Hours
+ Support and Sustain the TBDA () : void

 Non-Material Contributors 1..N:
+ Develop the TBDA Surveillance Mission()
+ Transport the TBDA()
+ Collect, Process and Assess Surveillance Information()
+ Disseminate the TBDA Information()
+ Support and Sustain the TBDA()
TBDA Development Options

Emphasis is still the Capability with options to pursue during concept evaluation and risk reduction phase (Post MS A)
JCIDS Process

- DOD Strategic Guidance
  - Joint Operations Concepts
    - Joint Operating Concepts
      - Joint Functional Concepts
      - Joint Integrating Concepts
  - Functional Area Analysis
    - Functional Needs Analysis
      - JCD
    - DOTLPF Analysis (Non-materiel Approaches)
    - Ideas for Materiel Approaches
    - Analysis of Materiel/Non-materiel Approaches
    - Alternative N
      - Alternative 2
      - Alternative 1
  - Integrated Architectures
  - Post Independent Analysis

*CJCSM 3170.01B*
Post Independent Analysis

- Final independent review of FAA, FNA, and FSA
  - Not the same people who conducted the analyses
  - Ensure...
    - Analyses were thorough
    - Potential solutions are reasonable
    - Potential solution set is complete
FAA, FNA, FSA, PIA Output

- There is no capability gap

- Capability gap can be addressed by change to:
  - DOT_LP - Doctrine, organization, training, leadership/education, personnel, and facilities
  - DCR - DOTLPF Change Request

- A materiel solution is required
  - Initial Capabilities Document (ICD)
Conclusion on Functional Analyses

- FAA, FNA, FSA, and PIA are import steps to identify, assess, and prioritize joint military capability needs
  - FAA – required capabilities
  - FNA – gaps in planned capabilities
  - FSA – potential solutions

- Conducted through a combination of quantitative and qualitative analyses

- Involve all stakeholders in the process
Tutorial Wrap-up

- JCIDS is an engineering intensive process!
- The Systems Engineering Method is appropriate for guiding the JCIDS analyses in every phase of the capability/system life cycle
  - Ensures traceability system functionality back to requirements
- Model Driven SE enables the JCIDS Team to fully understand what they are doing
  - SE Models provide the basis for the system’s architecture and all architectural views
  - SE Model is a living entity that transitions from JCIDS Team to Development Team
    - Today’s “To-Be” model becomes tomorrow’s “As-Is”
Tutorial Wrap-up

- JCIDS Functional Analyses, including AMA and PIA, are essential SE functions
  - Each phase, from FAA through FSA, better quantifies the degree of “materialization”
    - Including non-materiel capability contributors
- OMG SysML is most appropriate for modeling capabilities from early conceptualization to system design
  - Either OO or Traditional Structured methods

Good SE ↔ Effective JCIDS
Integrating Systems Engineering with Earned Value Management

NDIA Systems Engineering Conference
San Diego, CA
October 23, 2006

Paul J. Solomon, PMP
Performance-Based Earned Value®
Paul.Solomon@PB-EV.com

© 2006 Paul J. Solomon
Agenda

• Federal Policy and Guidance, Customer Expectations, EVM Limitations
• Newest Standards, Models, and Best Practices
• Project Management with Performance-Based Earned Value® (PBEV™)
• Implementing PBEV into Your Project
• IT/Software Progress Measurement Issues
• Implementing Better Acquisition Management into Your Project
Copyright © 2006 by Paul Solomon. This material contains excerpts from the book, *Performance-Based Earned Value.*® The excerpts were reprinted courtesy of John Wiley & Sons, Inc. and the IEEE Computer Society Press. *Performance-Based Earned Value*® was written by Paul Solomon and Ralph Young and copyrighted by the IEEE, 2007.
Project Management
Shortfalls

• Inadequate early warning
• Schedules, EV overstate true progress
• Remaining work underestimated
Does EVMS Really Integrate?

- **WBS**: Work Breakdown Structure
  - **COST**
  - **SCHEDULE**
  - **TECHNICAL PERFORMANCE**
  - **RISK**

© 2006 Paul J. Solomon
Value of Earned Value

EVM data will be reliable and accurate only if:

- The right base measures of technical performance are selected and
- Progress is objectively assessed.
Federal Policy and Guidance, Customer Expectations, EVM Limitations
Government Pays But Fails to Get Desired Outcomes

<table>
<thead>
<tr>
<th>GAO Report</th>
<th>Title</th>
<th>Findings and Recommendations</th>
</tr>
</thead>
<tbody>
<tr>
<td>06-66</td>
<td>Defense Acquisitions: DOD Paid Billions in Award and Incentive Fees</td>
<td>• Contractors not held accountable for achieving desired outcomes:</td>
</tr>
<tr>
<td></td>
<td>Regardless of Acquisition Outcomes</td>
<td>o Cost goals</td>
</tr>
<tr>
<td></td>
<td></td>
<td>o Schedule goals</td>
</tr>
<tr>
<td></td>
<td></td>
<td>o Desired capabilities</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Programs do not capture early on the requisite knowledge needed to effectively manage program risks</td>
</tr>
<tr>
<td>06-391</td>
<td>Defense Acquisitions: Assessments of Major Programs</td>
<td>DOD needs to change its requirements and budgeting processes to get desired outcomes from the acquisition process</td>
</tr>
</tbody>
</table>

(a) Government Accountability Office

© 2006 Paul J. Solomon
<table>
<thead>
<tr>
<th>GAO Report</th>
<th>Title</th>
<th>Findings and Recommendations</th>
</tr>
</thead>
<tbody>
<tr>
<td>06-250</td>
<td>Information Technology: Improve the Accuracy and Reliability of</td>
<td>2. If EVM not implemented effectively, decisions based on inaccurate</td>
</tr>
<tr>
<td></td>
<td>Investment Information</td>
<td>and potentially misleading information</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3. Agencies not measuring actual vs. expected performance in meeting IT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>performance goals.</td>
</tr>
</tbody>
</table>
## GAO Best Practices

<table>
<thead>
<tr>
<th>GAO Report</th>
<th>Title</th>
<th>Findings and Recommendations</th>
</tr>
</thead>
<tbody>
<tr>
<td>04-722</td>
<td>Information Technology: DOD's Acquisition Policies and Guidance</td>
<td><strong>Best Practices and Controls:</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Ensure that <em>requirements</em> are traceable, verifiable, and controlled.</td>
</tr>
<tr>
<td>06-215</td>
<td>DOD Systems Modernization</td>
<td>• Trace requirements to system design specifications and testing documents.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Continually measure an acquisition’s <em>performance</em>, cost, and schedule against <em>approved baselines</em>.</td>
</tr>
</tbody>
</table>
## GAO Best Practices

<table>
<thead>
<tr>
<th>GAO Report No.</th>
<th>Title</th>
<th>Findings and Recommendations</th>
</tr>
</thead>
<tbody>
<tr>
<td>06-110</td>
<td>Best Practices: Better Support of Weapons System „Needed to Improve Outcomes</td>
<td>Best Practice Controls:</td>
</tr>
<tr>
<td></td>
<td>Defense Acquisitions: Major Weapon Systems Continue to Experience Cost and Schedule Problems</td>
<td>• Complete subsystem and system design reviews</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Demonstrate with prototype that design meets requirements</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Agreement that drawings are complete and producible</td>
</tr>
</tbody>
</table>
U.S. Federal Policy on SE
### DOD Policy & Guidance on SE

<table>
<thead>
<tr>
<th>Policy for Systems Engineering in DOD Policy</th>
<th>2/20/04</th>
</tr>
</thead>
<tbody>
<tr>
<td>Defense Acquisition Guidebook (DAG)</td>
<td>10/8/04</td>
</tr>
<tr>
<td>Systems Engineering Plan Preparation Guide (SEP)</td>
<td>2/10/06</td>
</tr>
<tr>
<td>WBS Handbook, Mil-HDBK-881A (WBS)</td>
<td>7/30/05</td>
</tr>
<tr>
<td>Integrated Master Plan (IMP) &amp; Integrated Master Schedule Preparation &amp; Use Guide (IMS)</td>
<td>10/21/05</td>
</tr>
<tr>
<td>Risk Management Guide for DOD Acquisition (RISK)</td>
<td>Aug. 06</td>
</tr>
</tbody>
</table>

© 2006 Paul J. Solomon
DOD Policy on Award Fees (1)

- Link award fees to desired program outcomes
- Tie award fees to
  - Identifiable interim outcomes
  - Discrete events or milestones
  - Timely completion of:
    - Preliminary design review (PDR)
    - Critical design review (CDR)
  - Assessment of interim progress towards PDR, CDR
- Provisions explain how a contractor’s progress will be evaluated

1: OUSD (AT&L) Memo: Award Fee Contracts, 3/29/06

© 2006 Paul J. Solomon
### DOD Policy & Guides

<table>
<thead>
<tr>
<th>Policy or Guide (1 of 3)</th>
<th>Policy</th>
<th>DAG</th>
<th>SEP</th>
<th>WBS</th>
<th>IMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Develop SEP</td>
<td>P</td>
<td>4.2.3.2</td>
<td>1.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Technical reviews:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Event-driven timing</td>
<td>P</td>
<td>4.5.1</td>
<td>3.4.4</td>
<td>3.2.3.1</td>
<td>2.3, 3.3.2</td>
</tr>
<tr>
<td>• Success criteria</td>
<td>P</td>
<td>4.5.1</td>
<td>3.4.4</td>
<td>3.2.3.1</td>
<td></td>
</tr>
<tr>
<td>• Assess technical</td>
<td>4.5.1</td>
<td>3.4.4</td>
<td>3.2.3.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>maturity</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Integrate SEP with:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• IMP</td>
<td>4.5.1</td>
<td>3.4.5</td>
<td></td>
<td>1.2, 2.3</td>
<td></td>
</tr>
<tr>
<td>• IMS</td>
<td>4.5.1</td>
<td>3.4.5</td>
<td></td>
<td>1.2, 2.3</td>
<td></td>
</tr>
<tr>
<td>• Technical Performance</td>
<td>4.5.1</td>
<td>3.4.4</td>
<td>1.2, 2.3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Measures (TPM)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• EVM</td>
<td>4.5.1</td>
<td>3.4.5</td>
<td>1.2, 2.3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## DOD Guides

<table>
<thead>
<tr>
<th>Guide (2 of 3)</th>
<th>DAG</th>
<th>SEP</th>
<th>WBS</th>
<th>IMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integrate WBS with requirements specification, statements of work (SOW), IMP, IMS, and EVMS</td>
<td></td>
<td></td>
<td>2.2.3, 3.2.3.3</td>
<td>3.4.3</td>
</tr>
</tbody>
</table>
| TPMs to compare actual vs. plan:  
  • Technical development  
  • Design maturity | 4.5.5 | 3.4.4 |       | 3.3.2 |
| TPMs to report degree to which system requirements are met:  
  • Performance  
  • Cost  
  • Schedule | 4.5.5 | 3.4.4 |       |       |
| Standards and models to apply SE | 4.2.2 | 4.2.2.1 |       |       |
| Institute requirements management and traceability | 4.2.3.4 | 3.4.4 |       |       |
### Guides (3 of 3)

<table>
<thead>
<tr>
<th>Contractor:</th>
<th>IMS</th>
<th>Risk</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Incorporate <strong>risk mitigation</strong> activities into the <strong>IMS</strong> and <strong>budgets</strong></td>
<td>3.5</td>
<td>8.6.6</td>
</tr>
<tr>
<td>• Use <strong>IMS</strong> and <strong>EVM</strong> to monitor progress against <strong>risk plans</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Include **quantified risk assessments** in Estimate at Completion (EAC)

© 2006 Paul J. Solomon
OMB

• Office of Management and Budget (OMB)
• Circular No. A-11, Section 300
  • Planning, Budgeting, Acquisition and Management of Capital Assets
• Section 300-5
  • *Performance-based* acquisition management
  • Based on EVMS standard
  • Measure progress towards milestones
    • Cost
    • *Capability to meet specified requirements*
    • Timeliness
    • Quality
Newest Standards, Models, and Best Practices
Quality

Quality = technical performance:

*Ability* (current or projected)

of a set of inherent characteristics of a product

Product component or

Process

to *fulfill requirements* of customers

CMMI definition
SE Life Cycle Work Products
IEEE 1220

Requirements Analysis
- Requirements Baseline
- Requirements Validation
  - Validated Requirements Baseline
- Functional Analysis
  - Functional Architecture
  - Functional Verification
  - Verified Functional Architecture
- Synthesis
- Physical Architecture
- Design Verification
  - Verified Physical Architecture

Requirements trade studies and assessments
Functional trade studies and assessments
Design trade studies and assessments
## DOD Technical Baselines

<table>
<thead>
<tr>
<th>DAG Technical Review</th>
<th>DAG Baseline</th>
<th>DAG</th>
<th>IEEE 1220</th>
</tr>
</thead>
<tbody>
<tr>
<td>System Functional Review</td>
<td>System Functional Baseline</td>
<td>4.3.3.4.3</td>
<td>Validated Requirements Baseline</td>
</tr>
<tr>
<td>Preliminary Design Review</td>
<td>System Allocated Baseline</td>
<td>4.3.3.4.4</td>
<td>Verified Physical Architecture</td>
</tr>
<tr>
<td>Critical Design Review</td>
<td>System Product Baseline</td>
<td>4.3.3.4.5</td>
<td>Verified Physical Architecture</td>
</tr>
<tr>
<td>Production Readiness Review</td>
<td>System Product Baseline</td>
<td>4.3.3.9.3</td>
<td>Verified Physical Architecture</td>
</tr>
</tbody>
</table>
Performance-Based Progress Measurement

Measure the *allocated requirements* to determine:
- Development maturity vs. plan
- Indicated Quality
## Requirements Progress

<table>
<thead>
<tr>
<th>IEEE 1220</th>
<th>EIA-632</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.8.1.5 Performance-based progress measurement</td>
<td>4.2.1 Planning process, Req. 10: Progress against requirements</td>
</tr>
<tr>
<td>6.8.6 Track product ... metrics</td>
<td></td>
</tr>
<tr>
<td><strong>6.8.1.5 d) Assess</strong></td>
<td>Assess progress ...</td>
</tr>
<tr>
<td>• Development maturity to date</td>
<td>• Compare system definition</td>
</tr>
<tr>
<td>• Product’s ability to satisfy requirements</td>
<td>Against requirements</td>
</tr>
<tr>
<td>6.8.6 Product metrics...at pre-established</td>
<td>a) Identify product metrics and expected values</td>
</tr>
<tr>
<td>control points enable:</td>
<td>• Quality of product</td>
</tr>
<tr>
<td>• Overall system quality evaluation</td>
<td>• Progress towards satisfying requirements</td>
</tr>
<tr>
<td>• Comparison to planned goals and targets</td>
<td>D) Compare results against requirements</td>
</tr>
</tbody>
</table>
## Technical Performance Measures (TPM)

<table>
<thead>
<tr>
<th><strong>IEEE 1220: 6.8.1.5, Performance-based progress measurement</strong></th>
<th><strong>EIA-632: Glossary</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TPMs are key to progressively assess technical progress</strong></td>
<td><em>Predict</em> future value of <em>key technical parameters</em> of the end system based on current assessments</td>
</tr>
<tr>
<td>• Establish <em>dates</em> for</td>
<td><em>Planned value</em> profile is time-phased achievement projected</td>
</tr>
<tr>
<td>– Checking Progress</td>
<td>• <em>Achievement to date</em></td>
</tr>
<tr>
<td>– Meeting full conformance to requirements</td>
<td>• <em>Technical milestone where TPM evaluation is reported</em></td>
</tr>
</tbody>
</table>
• How well a system is achieving performance requirements
• Use actual or predicted values from:
  – Engineering measurements
  – Tests
  – Experiments
  – Prototypes
• Examples:
  – Payload
  – Response time
  – Range
  – Power
  – Weight
Use TPMs as a base measure of EV

TPM Planned Value Profile

Tolerance Bands

Achievement to Date

Percent Required Value

© 2006 Paul J. Solomon
## INCOSE Warning on TPM

**TPM per INCOSE Systems Engineering Handbook**

- **TPMs express the objective performance requirements.**

- **Without TPM**
  - Project manager could fall into the trap of relying on cost and schedule status alone
  - Can lead to a product developed on schedule and within cost that does not meet all key requirements.

- **Periodic recording of status of each TPM**
  - Provides continuing verification of degree of anticipated and actual achievement of technical parameters.
Success Criteria and Work Products Per SE Standards
Validated Requirements Baseline

IEEE 1220, (6.2): Success Criteria

- Represents identified customer expectations
- Represents constraints
  - Project
  - Enterprise
  - External
- Stays within constraints.
### Validated Requirements Baseline

**IEEE 1220, (6.1, 6.2): Work Products**

- Customer expectations
- Project, enterprise and external constraints
- Operational scenarios
- Measures of effectiveness (MOE)
- Interfaces
- Functional requirements
- Measures of performance (MOP)
- Modes of operation
- Design characteristics
- Human factors
- Documented trade-offs
Verified Functional Architecture

IEEE 1220, (6.3, 6.4): Work Products

- Functional context analysis
  - Functional behaviors
  - Functional interfaces
  - Allocated performance requirements
- Functional decomposition
  - Subfunctions
  - Subfunction states and modes
  - Data and control flows
  - Functional failure modes and effects
**IEEE 1220, (6.4): Success Criteria**

- Meets requirements of *validated requirements baseline*
- System functions decomposed to *lower-level functions* that shall be satisfied by elements of the system design
  - Subsystems
  - Components
  - Parts
- Requirements upwardly traceable to the validated requirements baseline
5.2.4.1 Subsystem reviews

a. Subsystem definition
  • Mature
    – Meet SE milestone criteria

a. Component allocations and specifications
  – Provide a sound subsystem concept

c. Subsystem risks assessed and mitigated

d. Trade-study data...substantiate that subsystem requirements are achievable
### Success Criteria of Technical Reviews

<table>
<thead>
<tr>
<th>IEEE 1220, 5.2 Preliminary design stage</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.2.4.2 System review</td>
</tr>
<tr>
<td>• After completion of subsystem reviews</td>
</tr>
<tr>
<td>• Does total system approach to detailed design satisfy the system baseline?</td>
</tr>
<tr>
<td>• Unacceptable risks are mitigated</td>
</tr>
<tr>
<td>• Issues for all subsystems, products, and life cycle processes are resolved</td>
</tr>
<tr>
<td>• Accomplishments and plans warrant continued development effort.</td>
</tr>
</tbody>
</table>
## IEEE 1220, Detailed design stage (Critical Design Review (CDR))

### 5.3.4.1 Component reviews

a. Detailed component definition... *mature*... meet

- MOE
- MOP criteria;

c. Risks... *mitigated* to... support fabrication, assembly, integration, test.

d. Trade-study data ... substantiate that detailed component requirements are achievable
5.3.4.3 System review

- After component and subsystem reviews
- Does detailed design *satisfy the system baseline?*
- Unacceptable *risks* are *mitigated*
- *Issues* for all subsystems, products, and life cycle processes are *resolved*
### IEEE 1220, (6.6): Success Criteria

- Design solution meets:
  - Allocated performance requirements
  - Functional performance requirements
  - Interface requirements
  - Workload limitations
  - Constraints
  - Use models and/or prototypes to determine success
### IEEE 1220, (6.6): Success Criteria (continued)

<table>
<thead>
<tr>
<th>• Design solution satisfies</th>
</tr>
</thead>
<tbody>
<tr>
<td>– Functional architecture</td>
</tr>
<tr>
<td>– Requirements baseline</td>
</tr>
<tr>
<td>– (Use models and/or prototypes)</td>
</tr>
<tr>
<td>• Requirements of the lowest level of the design architecture, including derived requirements, are traceable to the verified functional architecture.</td>
</tr>
</tbody>
</table>
Design Solution
Work Products

IEEE 1220, (6.5, 6.6): Work Products

- Integrated data package to document the selected design elements:
  - Drawings
  - Schematics
  - Software documentation
  - Manuals
  - Procedures
### Design Solution Work Products

<table>
<thead>
<tr>
<th>IEEE 1220, (6.5, 6.6): Work Products</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Design solution alternatives</td>
</tr>
<tr>
<td>• Physical interfaces</td>
</tr>
<tr>
<td>• Models and prototypes</td>
</tr>
<tr>
<td>• Failure modes and effects analyses (FMEA)</td>
</tr>
<tr>
<td>• Requirements traceability and allocation matrices</td>
</tr>
<tr>
<td>• Trade off analysis results</td>
</tr>
<tr>
<td>• Finalized design and description of interfaces</td>
</tr>
</tbody>
</table>
EVM Process Improvement through Capability Maturity Model Integration® (CMMI)
Product Requirements

- CMMI, PMBOK Guide: Traceability and consistency

Requirements

Product Requirements
Baseline

Work

- Project Plans
  - Task 1
  - Task 2
  - Task 3

- Activities
- Work Products
Measurement & Analysis

- CMMI Specific Practice (SP) 1.1:
  - Establish *measurement objectives* derived from *information needs* and *objectives*
- CMMI SP 1.2:
  - Specify *quantifiable* measures to address measurement objectives
    - Stated in *precise, unambiguous* terms
    - *Operational definitions* for the measures
    - Specify *how measurement data* will be obtained
- EVMS: “by *management assessment*”
Process and Product QA

- Product QA
  - CMMI:
    - *Objectively* evaluate work products against *clearly stated criteria*
    - *Minimize subjectivity*
  - EVMS:
    - EV is measurement of *quantity* of work
    - “*Quality* and *technical* content of work performed are *controlled by other means!*”
CMMI Typical Work Products

- Requirements Development
  - Product and product-component requirements
  - Interface requirements
  - Required functionality
  - Product component operational concepts, scenarios and environments
  - TPMs
CMMI Typical Work Products

- Technical Solution
  - Operational concepts and scenarios
  - Technical data package
    - Allocated requirements
    - Product-component descriptions
    - Key product characteristics
    - Required physical characteristics and constraints
  - Interface requirements
  - Material requirements
  - Verification criteria to achieve requirements
CMMI Typical Work Products

- Technical Solution
  - Comprehensive product-component interface
    - Interface design specs.
    - Interface control documents
    - Interface specification criteria
  - Implemented design
CMMI Typical Work Products

- Requirements management
  - Requirements traceability matrix (RTM)
- Verification
  - Exit and entry criteria for work products
  - Verification results
- Measurement and analysis
  - Specifications of base and derived measures
- Decision analysis and resolution
  - Results of evaluating alternative solutions (trade-studies)
PMBOK® Guide

PMBOK®, Guide to Project Management
PMBOK® Guide (5.5).

- **Product scope**
  - Features and functions that characterize a
    - Product
    - Service
    - Result
- **Project scope**
  - Work that needs to be accomplished to deliver a
    - Product, service or result
    with the specified features and functions.
PMBOK Guide®: Quality

- Establish a quality baseline as part of the PMB (8.1.3.5)
  - Integrate technical and quality objectives (10.3.1.5)
- Specify TPMs to measure schedule performance (11.6.2.4)
4. Project Management with Performance-Based Earned Value® (PBEV℠)
• 4 Principles and 16 Guidelines
• Specify most effective measures of project performance
• Requirements-driven plan
• Consistent with standards and models
• Tailorable and scalable, depending on risk
• Lean
PBEV and Quality

• Link EV with evolving development maturity or **quality**

• **Quantify** quality measures
  – Percent of product requirements met (weighted)
  – Technical performance achieved

• **Measure quality**
  – Of “completed” work products
  – Of work in process

**EV without Quality has less management value**
PBEV Based on Standards and Models

- ANSI/EIA-632
- IEEE 1220
- CMMI®
- PMBOK® Guide
- INCOSE SE Handbook
- PSM. Practical Software and Systems Measurement: A Foundation for Objective Project Management
• Integrates SE with EVM
  – Planning:
    • Link performance measurement baseline (PMB) to:
      – Product requirements (technical) baseline
      – SEP
      – SE process work products
    • Identify product metrics for performance-based progress measurement
      – Planned value profile of TPMs
      – Planned development maturity to date
    • Success criteria (reviews and work products)
PBEV Characteristics

• Integrates SE with EVM
  – Measurement
    • Objective measurement of interim progress
    • Progress of requirements through engineering life cycle
    • EV linked with
      – Indicated quality of end product
      – TPM achieved
  • EV used to measure Quality
    – Not just work accomplished
PBEV Characteristics

- Meaningful analysis
  - Correlate analyses of deviations from plan:
    - Technical maturity/quality
    - Schedule
    - Cost
PBEV Characteristics

- Lean
  - Minimizes costs; measurement costs money
  - Fewer work packages with right base measures
    - Product requirements-driven
    - Quality measures
    - Work products
- Applicable to all development models and methods
  - Waterfall, incremental, spiral, V, evolutionary, agile
Principles of PBEV

1. Integrate product requirements and quality into the project plan.
2. Specify performance towards meeting product requirements, including planned quality, as a base measure of earned value.
3. Integrate risk management with EVM.
4. Tailor the application of PBEV according to the risk.
Supplemental PBEV Process Flow

(P) Establish product requirements and components (technical baseline)

Guideline 1.1

(1) Establish product requirements and components (technical baseline)

Guidelines 1.2, 2.2

Plan the work (Schedule & Budget)

(P) Integrate product requirements and quality with plan

Guidelines 3.2, 3.2, 4.1, 4.2

Execute the plan

(P) Integrate risk management with plan

Guideline 2.7

Measure the work

(P) Measure progress towards meeting product requirements and quality

Incorporate internal/external changes

Analyze variances

Implement corrective action

(P) = Supplemental PBEV Process

© 2006 Paul J. Solomon
1.1 Establish *product requirements* and allocate these to product components.

1.2 Maintain *bidirectional traceability of product* and product component *requirements among:*  
   – Project plans  
   – Work packages and planning packages  
   – Work products.
1.3 Identify *changes* that need to be made to
• Project plans
• Work packages
• Planning packages
• Work products *resulting from changes to the product requirements.*

2.1 *Define the information need and objective* to measure *progress* towards satisfying *product requirements.*
2.2 Specify *work products* and performance-based *measures* of progress for meeting *product requirements* as *base measures of earned value*. 
2.3 Specify *operational definitions* for the base measures of EV, stated in *precise, unambiguous terms*

Address:

- Communication
  - What has been measured
  - How it was measured
  - What are the units of measure
  - What has been included or excluded

Repeatability: can the measurement be repeated, given the same definition, to get the same results?
2.4 Identify **event-based success criteria** for technical reviews:

- *Development maturity to date*
- *Product’s ability to meet product requirements.*
2.5 Establish:

- Time-phased, *planned values* for measures of *progress towards meeting product requirements*
- Dates or frequency for checking progress
- Dates when *full conformance will be met*.

2.6 Allocate budget in discrete work packages to measures of progress towards meeting *product requirements*. 
2.7 Compare

• Amount of planned budget and
• Amount of budget earned

for achieving progress towards meeting *product requirements*
2.8 Use the level of effort (LOE) method to plan work that is measurable but is not a measure of progress towards meeting
• Product requirements
• Final cost objectives
• Final schedule objectives.

2.9 Perform more effective variance analysis by segregating discrete effort from LOE.
3.1 Identify *changes* that need to be made to
- Project plans
- Work packages
- Planning packages
- Work products
  resulting from *responses to risks*.

3.2 Develop revised EAC
based on *risk quantification*
4.1 Tailor the application of PBEV to the elements of the WBS according to the risk.

4.2 Tailor the application of PBEV to the phases of the system development life cycle according to the risk.
Requirements Development and Management
Manage Requirements

- Second most critical requirements practice
- Example: Use an additional radio band width.
- Changes to plan
  - Trade studies to determine best solution.
  - Budget and schedule changed.
  - All subsequent milestones moved to right
  - Higher cost to customer caused by
    - Level of effort activities extended
    - Skill retention (of people on discrete tasks).
Trade Studies

• Provide objective foundation to select an approach to the solution of an engineering problem.

• Typical trade results:
  • Select user/operational concept
  • Select system architectures
  • Derive requirements
    • Alternative functional approaches to meet requirements
  • Requirements allocations
  • Technical/design solutions
  • Cost analysis results
  • Risk analysis results

© 2006 Paul J. Solomon
Maintaining the Technical Requirements Baseline

• Baseline
  – Specification or product that has been formally reviewed and agreed on.
  – Serves as the basis for further development.
  – Changed only through formal change control procedures.

• Maintaining *product* requirements baseline supports planning and control.
Good SE planning:
• Manage the technical baselines
• Technical baselines:
  • Are specific SE work products
  • Provide product-driven view for SE cost management
• Maturity of baselines are entry criteria for event-based technical reviews
• EV provides critical insight to technical progress

Source: Office of the Undersecretary of Defense (Acquisition, Technology & Logistics)/Defense Systems website
Risk Management
Risk vs. Issue

Source: Risk Management Guide for DOD Acquisition:

- If root cause is described in past tense, it has already occurred. It is an issue.

- Incorporate risk mitigation activities into the IMS and EVM
  - Monitor progress against risk plans

- Include quantified risk impacts in EAC
### EVMS: Not a Risk Management Tool

<table>
<thead>
<tr>
<th>Significant Variance</th>
<th>Issue?</th>
<th>Risk?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cost</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>Schedule</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

© 2006 Paul J. Solomon
Implementing PBEV into Your Project
PBEV Techniques

• Allocate budget to completion of
  – Enabling work products (drawings, code)
  – Allocated requirements
• Establish milestones with success criteria
  – SE Life Cycle Work Products
  – Number of allocated requirements to be met
  – Technical performance
    • Planned technical maturity
    • Quality
PBEV Techniques

• Measure quality
  – Work products (partial and complete)
  – Technical maturity of evolving product
  – Use analysis, models, simulations, prototypes

• Base EV on
  – Work products (drawings, code) and
  – Quality
PBEV Techniques

• Use LOE if work is measurable but is *not* a measure of progress towards meeting:
  – Product requirements
  – Final cost objectives
  – Final schedule objectives
Initial Design Development Measures

• Design (work unit progress):
  – Base EV on
    • # Enabling work products and
      # Requirements met
  – Example:
    • # Components designs completed
      and
    • # Requirements met traced to components

- Recommended PBEV Measure
EX 1: EV Based on Drawings and Requirements

- **SOW:** Design a subsystem with 2 TPM requirements:
  - Maximum (Max.) weight: 200 lb.
  - Max. diameter: 1 inch

- **Enabling work products:** 50 drawings

- **BAC:** 2000 hours
  - Drawings: 40 hours/drawing @ 50 2000
  - Requirements *not* met on schedule:
    - Potential **negative** EV
      - Weight: -100
      - Diameter -200
EX 1: Schedule Plan and Status

Status at April 30
- Drawings completed: 41
- Weight requirement *not* met
- Diameter requirement met
EX 1: Earned Value

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Planned drawings cur</td>
<td>8</td>
<td>10</td>
<td>12</td>
<td>10</td>
<td>10</td>
<td>50</td>
</tr>
<tr>
<td>Planned drawings cum</td>
<td>8</td>
<td>18</td>
<td>30</td>
<td>40</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>BCWS cur</td>
<td>320</td>
<td>400</td>
<td>480</td>
<td>400</td>
<td>400</td>
<td>2000</td>
</tr>
<tr>
<td>BCWS cum</td>
<td>320</td>
<td>720</td>
<td>1200</td>
<td>1600</td>
<td>2000</td>
<td>2000</td>
</tr>
<tr>
<td>Actual drawings cur</td>
<td>9</td>
<td>10</td>
<td>10</td>
<td>12</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>Actual drawings cum</td>
<td>9</td>
<td>19</td>
<td>29</td>
<td>41</td>
<td>49</td>
<td></td>
</tr>
<tr>
<td>EV (drawings) cum</td>
<td>360</td>
<td>760</td>
<td>1160</td>
<td>1640</td>
<td>1960</td>
<td></td>
</tr>
<tr>
<td>Negative EV Reqs cum</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>-100</td>
<td></td>
</tr>
<tr>
<td>Net EV cum</td>
<td>360</td>
<td>760</td>
<td>1160</td>
<td>1640</td>
<td></td>
<td>-1860</td>
</tr>
</tbody>
</table>

SV = -140
EX 1: Variance Analysis

Variance analysis (drawings and requirements):

- 1 drawing behind schedule          - 40
- Diameter requirement met           - 0
- Weight requirement *not* met:       - 100

Schedule variance                   - 140
TPM at Higher WBS Level

• Design of a component at the work package level
• Completion of the comp. design depends on
  – Achieving allocated TPMs values at
    1. Component level and
    2. Subsystem level
• EV is dependent on planned TPM values achieved at both levels
TPM at Higher WBS Level

- For a weight TPM, all components play a part
- For other TPMs, such as response time
  - Subsets of the components combine to meet subsystem performance objectives
    - Hardware components
    - Software components
TPM at Higher Level

• Assumptions:
  – Component in Example 3 is one of four components that form a subsystem
  – Subsystem’s TPM objective is 4000 lb.
  – SEP states:
    Some components may be overweight at completion if there are offsets in other components (Comp)
    as long as the total subsystem (Sub) weight does not exceed 4000 lb.
## EX 2: TPM at Higher WBS Level

<table>
<thead>
<tr>
<th>Component, Work Pkg.</th>
<th>TPM Planned Value</th>
<th>Planned Completion</th>
<th>Component EV Penalty</th>
<th>Subsystem EV Penalty</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>200</td>
<td>April</td>
<td>-100</td>
<td>-50</td>
</tr>
<tr>
<td>2</td>
<td>1000</td>
<td>April</td>
<td>-500</td>
<td>-250</td>
</tr>
<tr>
<td>3</td>
<td>2000</td>
<td>May</td>
<td>-1000</td>
<td>-500</td>
</tr>
<tr>
<td>4</td>
<td>800</td>
<td>May</td>
<td>-400</td>
<td>-200</td>
</tr>
<tr>
<td><strong>Subsystem total</strong></td>
<td><strong>4000</strong></td>
<td></td>
<td><strong>-2000</strong></td>
<td><strong>-1000</strong></td>
</tr>
</tbody>
</table>
### TPM at Higher WBS Level

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Planned drawings cur</td>
<td>8</td>
<td>10</td>
<td>12</td>
<td>10</td>
<td>10</td>
<td>50</td>
</tr>
<tr>
<td>Planned drawings cum</td>
<td>8</td>
<td>18</td>
<td>30</td>
<td>40</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>BCWS cur</td>
<td>320</td>
<td>400</td>
<td>480</td>
<td>400</td>
<td>400</td>
<td>2000</td>
</tr>
<tr>
<td>BCWS cum</td>
<td>320</td>
<td>720</td>
<td>1200</td>
<td>1600</td>
<td>2000</td>
<td>2000</td>
</tr>
<tr>
<td>Actual drawings completed cur</td>
<td>9</td>
<td>10</td>
<td>10</td>
<td>12</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>Actual drawings completed cum</td>
<td>9</td>
<td>19</td>
<td>29</td>
<td>41</td>
<td>49</td>
<td></td>
</tr>
<tr>
<td>EV (drawings) cum</td>
<td>360</td>
<td>760</td>
<td>1160</td>
<td>1640</td>
<td>1960</td>
<td></td>
</tr>
<tr>
<td>Negative EV Reqs cum</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>-1500</td>
<td></td>
</tr>
<tr>
<td>Net EV cum</td>
<td>360</td>
<td>760</td>
<td>1160</td>
<td>1640</td>
<td>460</td>
<td></td>
</tr>
</tbody>
</table>
# Ex. 3: Rework in Same WP

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Planned drawings –cur.</td>
<td>8</td>
<td>10</td>
<td>12</td>
<td>10</td>
<td>10</td>
<td>50</td>
</tr>
<tr>
<td>Planned drawings –cum.</td>
<td>8</td>
<td>18</td>
<td>30</td>
<td>40</td>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td>BCWS – cum.</td>
<td>320</td>
<td>720</td>
<td>1200</td>
<td>1600</td>
<td>2000</td>
<td>2000</td>
</tr>
<tr>
<td>Drawings completed</td>
<td>9</td>
<td>10</td>
<td>10</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Drawings returned</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>-5</td>
<td></td>
</tr>
<tr>
<td>Net drawings – cur.</td>
<td>9</td>
<td>10</td>
<td>10</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Net drawings – cum.</td>
<td>9</td>
<td>19</td>
<td>29</td>
<td>28</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Net EV – cur.</td>
<td>360</td>
<td>400</td>
<td>400</td>
<td>-40</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EV – cum.</td>
<td>360</td>
<td>760</td>
<td>1160</td>
<td>1120</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SV – cum.</td>
<td>0</td>
<td>40</td>
<td>-40</td>
<td>-480</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Drawings Returned for Rework Result in Negative EV*
## Ex 4: Rework in Separate WP

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Planned number of drawings</td>
<td>8</td>
<td>10</td>
<td>12</td>
<td>10</td>
<td>10</td>
<td>50</td>
</tr>
<tr>
<td>BCWS</td>
<td>288</td>
<td>360</td>
<td>432</td>
<td>360</td>
<td>360</td>
<td>1800</td>
</tr>
</tbody>
</table>

**Planned rework in WP 2**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>BCWS</td>
<td>60</td>
<td>70</td>
<td>70</td>
<td></td>
<td></td>
<td>200</td>
</tr>
</tbody>
</table>
Ex 4: Rework in Separate WP

<table>
<thead>
<tr>
<th>WP 2: Rework of Drawings</th>
<th>Jan.</th>
<th>Feb.</th>
<th>Mar. MS 1</th>
<th>Apr. MS 2</th>
<th>May MS 3</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>BCWS</td>
<td></td>
<td></td>
<td>60</td>
<td>70</td>
<td>70</td>
<td>200</td>
</tr>
</tbody>
</table>

Rework Milestones:

- Milestone 3: 100% of drawings meet requirements
- Milestone 2: 90% of drawings meet requirements
- Milestone 1: 80% of drawings meet requirements
EX 5: Trade Study

• Outcome is usually a recommendation that is needed to make a decision.
• Decision constrains and guides further progress.
• Work product: documented trade study results.
• Engineering processes should include a process and structured approach for performing trade studies.
  – Process should include both interim and final work products that can be:
    • Planned, scheduled, and measured.
Trade Study Outline

1. Purpose of Study:
   – Resolve an issue
   – Perform decision analysis
   – Perform analysis of alternatives

2. Scope of study
   – State level of detail of study
   – State assumptions
   – Identify influencing requirements and constraints.
3. Trade study description
Describe trade studies to be performed to make tradeoffs among:
– Concepts
– User requirements
– System architectures
– Design
– Program schedule
– Functional performance requirements
– Life-cycle costs
4. Analytical approach
   - Identify candidate solutions
   - Measure performance
   - Develop models and measures of merit
   - Develop values for viable candidates
   - Selection criteria: (normally risk, performance, cost)
5. Scoring
   - Determine measures of results to be compared to criteria
   - Assign weights to measures of results reflecting their relative importance
   - Perform sensitivity analysis

6. Evaluate alternatives

7. Documentation of trade results
## Trade Study Schedule

<table>
<thead>
<tr>
<th>Trade Study Base Measures: Evaluate Alternatives</th>
<th>Time Period</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial evaluation of each of 5 candidates has three milestones:</td>
<td></td>
</tr>
<tr>
<td>• Start test set up</td>
<td>1</td>
</tr>
<tr>
<td>• Tests executed to completion</td>
<td>2</td>
</tr>
<tr>
<td>• Analyze and document</td>
<td>3</td>
</tr>
<tr>
<td>Down select from 5 candidates to 2 candidates</td>
<td></td>
</tr>
<tr>
<td>Document recommendation</td>
<td>4</td>
</tr>
</tbody>
</table>
Trade Study EV

- Evaluation activity planning assumptions
- Total Budget: 1000 hours
  - Test and evaluate 5 candidates: 500
    - 100 per candidate
    - Take EV even if candidate discarded before test complete
  - Down select to 2 candidates: 200
  - Document final recommendation: 300
- Period of Performance: 4 months
EX 6: Requirements Management

• Discretely measure requirements management
• Use RTM to control plan
• Requirements management (RM) tasks
  – Defined
  – Validated
  – Determined verification method
  – Approved
  – Allocated
  – Traced to verification document (test procedure)
  – Tested
  – Verified
• Key indicator of project performance
– # requirements traced to software or hardware components

Note: Budget per Work Unit does not have to be equally distributed

- Recommended PBEV Measure
## Budget Allocation

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Budget %</td>
<td></td>
<td></td>
<td>15%</td>
<td>15%</td>
<td>15%</td>
<td>20%</td>
<td>15%</td>
<td>20%</td>
<td>100%</td>
</tr>
<tr>
<td>Component</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Enclosure</td>
<td>3</td>
<td>240</td>
<td>36</td>
<td>36</td>
<td>36</td>
<td>48</td>
<td>36</td>
<td>48</td>
<td>240</td>
</tr>
<tr>
<td>Transmitter</td>
<td>1</td>
<td>80</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>16</td>
<td>12</td>
<td>16</td>
<td>80</td>
</tr>
<tr>
<td>Battery</td>
<td>2</td>
<td>160</td>
<td>24</td>
<td>24</td>
<td>24</td>
<td>32</td>
<td>24</td>
<td>32</td>
<td>160</td>
</tr>
<tr>
<td>Control</td>
<td>1</td>
<td>80</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>16</td>
<td>12</td>
<td>16</td>
<td>80</td>
</tr>
<tr>
<td>Software</td>
<td>9</td>
<td>720</td>
<td>108</td>
<td>108</td>
<td>108</td>
<td>144</td>
<td>108</td>
<td>144</td>
<td>720</td>
</tr>
<tr>
<td>Total</td>
<td>16</td>
<td>1280</td>
<td>192</td>
<td>192</td>
<td>192</td>
<td>256</td>
<td>192</td>
<td>256</td>
<td>1280</td>
</tr>
</tbody>
</table>
## Time-Phased Budget

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Defined</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>Validated</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Verif. Method</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Allocated</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>Traced to Verif.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>Verified</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Defined</td>
<td>12</td>
<td>36</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>36</td>
</tr>
<tr>
<td>Validated</td>
<td>12</td>
<td>24</td>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>36</td>
</tr>
<tr>
<td>Verif. Method</td>
<td>12</td>
<td>12</td>
<td>24</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>36</td>
</tr>
<tr>
<td>Allocated</td>
<td>16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>48</td>
</tr>
<tr>
<td>Traced to Verif.</td>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>36</td>
</tr>
<tr>
<td>Verified</td>
<td>16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>48</td>
</tr>
<tr>
<td>Total</td>
<td>36</td>
<td>24</td>
<td>24</td>
<td>24</td>
<td></td>
<td>48</td>
<td>36</td>
<td>48</td>
</tr>
</tbody>
</table>

| BCWS cumulative            | 36   | 60   | 84   | 108  | 156 | 192  | 240  |       |

© 2006 Paul J. Solomon
## Earned Value

<table>
<thead>
<tr>
<th>Enclosure</th>
<th>Jan</th>
<th>Feb</th>
<th>Mar</th>
<th>Apr</th>
<th>May</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Completed</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Budget/Activity</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Defined</td>
<td>12</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Validated</td>
<td>12</td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Verif. Method</td>
<td>12</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td><strong>EV cumulative</strong></td>
<td></td>
<td>0</td>
<td>36</td>
<td>36</td>
<td>60</td>
</tr>
<tr>
<td><strong>BCWS cumulative</strong></td>
<td></td>
<td>36</td>
<td>60</td>
<td>84</td>
<td>108</td>
</tr>
<tr>
<td><strong>Schedule Variance</strong></td>
<td></td>
<td>-36</td>
<td>-24</td>
<td>-48</td>
<td>-48</td>
</tr>
</tbody>
</table>

© 2006 Paul J. Solomon
EX 7: PBEV Variance Analysis

- Negative EV causes sudden schedule and cost variances
- Example: New schedule variance when TPM planned value not achieved.
  - Requirements for control console:
    - Maximum surrounding temperature (Max.) < 100 degrees F. for more than 30 seconds
    - Max. never > 120 degrees
  - Prior status
    - Designs on schedule
      - Control console
      - Nearby equipment
      - Cooling methods
    - Thermal analyses on schedule
    - Meets TPM planned values at lower WBS levels
PBEV Variance Analysis

- Known performance issue
  - TPM planned value not achieved
  - Max. > 120 degrees
- Negative EV results in significant schedule variance
- How to describe in variance analysis?
PBEV Variance Analysis

• Cause:
  – Insufficient space between surrounding components
  – Insufficient airflow to cool the equipment
  – Root cause:
    • Requirements did not limit dimensions of cables and connectors
• Impact:
  – 4 week delay for redesign
  – Cost increase of & 50 K for redesign, retest
• Corrective Action Plan:
  – Rework requirements, design, test
  – Improve requirements development and validation process
PBEV EAC Tip

- If significant technical issues exist, only detailed planning can provide reliable EAC
- If significant risks exist (high probability and cost impact), include cost impact in EAC
IT/Software Progress Measurement Issues

© 2006 Paul J. Solomon
Initial Development: Incremental Capability

- Document baseline content of incremental builds
  - # functional requirements
  - # components
- Baseline the build milestones and completion criteria
- Baseline the build work packages and EV metrics
- Take EV based on functionality achieved
  - Show completed milestones & take full earned value
    **Only if all** completion criteria and planned functionality attained
Internal Replanning of Deferred Functionality

- If build is released short of planned functionality:
  - Take *partial* EV and leave work package open
  - Take *partial* EV and close work package
    - Transfer deferred scope and budget to first month of work package for next incremental build
      - EV mirrors technical performance
      - Schedule variance retained
    - Disclose shortfall and slips on higher schedules
EX 8: Deferred Functionality

- **SOW: Software Requirements in 2 Builds:**

<table>
<thead>
<tr>
<th>Build</th>
<th>Allocated Req.</th>
<th>Budget/Req.</th>
<th>BAC</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>100</td>
<td>5</td>
<td>500</td>
</tr>
<tr>
<td>B</td>
<td>60</td>
<td>5</td>
<td>300</td>
</tr>
</tbody>
</table>
### SW Build Plan

<table>
<thead>
<tr>
<th></th>
<th>Jan</th>
<th>Feb</th>
<th>Mar</th>
<th>Apr</th>
<th>May</th>
<th>Jun</th>
<th>Jul</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Build A</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Planned Reqs. met</td>
<td>25</td>
<td>25</td>
<td>25</td>
<td>25</td>
<td></td>
<td></td>
<td></td>
<td>100</td>
</tr>
<tr>
<td>Budget/Req.: 5 hours</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BCWS current (cur)</td>
<td>125</td>
<td>125</td>
<td>125</td>
<td>125</td>
<td></td>
<td></td>
<td></td>
<td>500</td>
</tr>
<tr>
<td>BCWS cumulative (cum)</td>
<td>125</td>
<td>250</td>
<td>375</td>
<td>500</td>
<td></td>
<td></td>
<td></td>
<td>500</td>
</tr>
<tr>
<td><strong>Build B</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Planned Reqs. Met</td>
<td></td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>60</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BCWS cur</td>
<td></td>
<td>100</td>
<td>100</td>
<td>100</td>
<td></td>
<td>300</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Deferred Functionality Status

<table>
<thead>
<tr>
<th>Build A</th>
<th>Jan</th>
<th>Feb</th>
<th>Mar</th>
<th>Apr</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Planned Reqs. Met cur</td>
<td>25</td>
<td>25</td>
<td>25</td>
<td>25</td>
<td>100</td>
</tr>
<tr>
<td>Actual Reqs. Met cur</td>
<td>20</td>
<td>20</td>
<td>25</td>
<td>25</td>
<td>90</td>
</tr>
<tr>
<td>BCWS cur</td>
<td>125</td>
<td>125</td>
<td>125</td>
<td>125</td>
<td>500</td>
</tr>
<tr>
<td>EV cur</td>
<td>100</td>
<td>100</td>
<td>125</td>
<td>125</td>
<td>450</td>
</tr>
<tr>
<td>BCWS cum</td>
<td>125</td>
<td>250</td>
<td>375</td>
<td>500</td>
<td></td>
</tr>
<tr>
<td>EV cum</td>
<td>100</td>
<td>200</td>
<td>325</td>
<td>450</td>
<td></td>
</tr>
</tbody>
</table>

**Schedule variance (SV):**

<table>
<thead>
<tr>
<th>Reqs. Met</th>
<th>SV</th>
</tr>
</thead>
<tbody>
<tr>
<td>-5</td>
<td>-25</td>
</tr>
<tr>
<td>-10</td>
<td>-50</td>
</tr>
<tr>
<td>-10</td>
<td>-50</td>
</tr>
<tr>
<td>-10</td>
<td>-50</td>
</tr>
</tbody>
</table>

**Total:** -50
# Deferred Functionality Replan

<table>
<thead>
<tr>
<th></th>
<th>Apr</th>
<th>May</th>
<th>Jun</th>
<th>Jul</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Close Build A work package</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>SV:</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reqs. Not Met</td>
<td>-10</td>
<td></td>
<td></td>
<td></td>
<td>-10</td>
</tr>
<tr>
<td>BCWS remaining</td>
<td>-50</td>
<td></td>
<td></td>
<td></td>
<td>-50</td>
</tr>
<tr>
<td><strong>Build B</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Before Replan:</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Planned Reqs. Met</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>60</td>
<td></td>
</tr>
<tr>
<td>BCWS cur</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>300</td>
<td></td>
</tr>
<tr>
<td><strong>Plus transfer:</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reqs.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>+10</td>
</tr>
<tr>
<td>BCWS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>+50</td>
</tr>
</tbody>
</table>

**After replan:**

<table>
<thead>
<tr>
<th></th>
<th>Apr</th>
<th>May</th>
<th>Jun</th>
<th>Jul</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Planned Reqs. Met</strong></td>
<td>30</td>
<td>20</td>
<td>20</td>
<td>70</td>
<td></td>
</tr>
<tr>
<td><strong>BCWS cur</strong></td>
<td>150</td>
<td>100</td>
<td>100</td>
<td>350</td>
<td></td>
</tr>
</tbody>
</table>
# Deferred Functionality Status

<table>
<thead>
<tr>
<th></th>
<th>May</th>
<th>Jun</th>
<th>Jul</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Build B After Replan:</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Planned Req. Met</td>
<td>30</td>
<td>20</td>
<td>20</td>
<td>70</td>
</tr>
<tr>
<td>BCWS cur</td>
<td>150</td>
<td>100</td>
<td>100</td>
<td>350</td>
</tr>
<tr>
<td><strong>Actual Req. Met cur</strong></td>
<td>20</td>
<td></td>
<td></td>
<td>20</td>
</tr>
<tr>
<td>EV cur</td>
<td>100</td>
<td></td>
<td></td>
<td>100</td>
</tr>
<tr>
<td><strong>Schedule variance cum:</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reqs. Met</td>
<td>-10</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SV</td>
<td>-50</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Software Quality Measures

- Software quality measures are TPMs
  - Defect density
  - Number of problem reports
- Failure to achieve planned quality indicates
  - More rework during development
  - More problems after product delivery
Ex 9: Quality as a Limiter of EV

- TPM: Number or density of defects
- If defect plan/quality goals are not attained: Use Hammervold Algorithm:
  - Assumes that more defects than planned will be found in future
  - Assumes that some verified requirements will be negated by future defects
  - Limits EV to 80% of BAC even if % requirements met > 80%
Implementing Better Acquisition Management into Your Project
Acquisition Management

Ensure Contractors Integrate SE with EVM

• Requirements, incentives, insight:
  – Solicitation/Request for Proposal (RFP)
  – Integrated Master Plan (IMP)
  – Integrated Baseline Review (IBR)
  – Integrated Master Schedule (IMS)
  – EVMS compliance assessments
  – Independent technical assessments
  – Monitor consistency and validity of reports
  – Independent EAC and risk assessments
  – Award fee criteria
• Event-based technical reviews
  – Appropriate entry and exit criteria
• Show build up from unit design and test to subsystem to system integration
• Software development and integration approach reflected in significant accomplishments and criteria
IBR

- Review implementation of SE:
  - Entry and success criteria for IMP events
  - Requirements management and traceability
  - Control points for product metrics and TPMs
  - Milestones with technical maturity success criteria
    - TPM planned values
    - Meeting requirements
    - Percent of designs complete
  - SE life cycle work products in IMS
IBR

- Confirm integration of
  - Technical baseline
  - WBS
  - IMP/IMS
  - EVM
  - Risk management
IMP > IMS Flowdown

• Meeting all the IMP criteria indicates completion of the significant accomplishment.
• The IMS should decompose IMP criteria into tasks necessary to meet the criteria.
IMS Checklist

• Review completion criteria for significant accomplishments
  • Objective and measurable
• Review each significant accomplishment
  • Do events occur at the system level and across multiple Integrated Product Teams (IPT)?
  • Does each criterion relate to a specific IPT?
IMS Checklist

- Subcontracted efforts
  - Appropriate visibility?
    - Requirements flowdown
    - Design reviews prior to system level reviews
- IMS traceable to EVMS work and planning packages
  - Timing
  - Completion criteria
  - WBS numbers
Independent Technical Assessments

• Verify technical maturity
  – Product and Quality metrics
    • TPM achievement
    • Requirements met
Monitor Consistency and Validity of Reports

• Compare performance reports for consistency:
  – Program status
    • Technical
    • Schedule
    • EV
  – Variance analyses
    • Root causes
    • Corrective action plans
    • Impacts on cost and schedule
Independent Assessments of EAC and Risks

- Perform EAC and quantified risk assessments
  - At total contract level
  - At lower WBS levels
- Compare your assessment with supplier’s
- Resolve significant differences
- Validate supplier’s corrective action plans
  - Performance efficiency
  - Schedule recovery
Award Fee Criteria for Successful I BR

• Agree on:
  – Entry and exit criteria for event-based technical reviews
  – Objective completion criteria for each significant accomplishment needed to support reviews.
• Subcontracted efforts on IMS have sufficient:
  – Milestones
  – Accomplishments
  – Completion criteria
• Completions of subcontractor design reviews occur prior to system level design reviews.
Award Fee Criteria for Successful IBR

- Management process provides effective:
  - Integrated technical/schedule/cost planning
  - Baseline control
- Valid critical path
- Technical baselines are included in the IMS
- TPM milestones are in IMS with planned values
- IMS milestones and completion criteria traceable to EVMS work and planning packages
Award Fee Criteria for Successful IBR

- Process, bidirectional traceability among:
  - Requirements
  - Work products
  - Project plans (IMS, work and planning packages)
- IMS includes activities identified in risk mitigation
- PMB is sufficient to successfully execute the project
Award Fee Criteria for Successful Technical Reviews

- All exit criteria for event-driven technical reviews met on schedule
  - Development maturity is on schedule
  - Issues resolved
    - All subsystems
    - Products
    - Life cycle processes
- Unacceptable risks are mitigated
Award Fee Criteria for Successful Technical Reviews

- System design is capable of meeting requirements
- Cost performance objectives have been met
- Bidirectional traceability is maintained among the requirements and the project plans, work products, and work packages.
- Accomplishments and plans satisfy criteria for continuation of the technical effort
Summary

• Integrate
  – Systems engineering with PBEV
    • Product requirements
    • Manage the technical baseline
    • Technical performance measures
    • SE life cycle work products
      – Technical>schedule>cost performance
  • Lean process
    – Less work packages with right base measures
• Agile

© 2006 Paul J. Solomon
Benefits of Using PBEV

- Management Decisions
- Identify Problem Areas
- Take Corrective Action
- Change Future Direction
- Forecast Future Performance

Data Analysis

EVMS Input

- Budget
- Schedule
- EV
- Actual Costs

Technical Requirements / Quality

Risk Management Output

EVMS Output
Process Improvement

But wait.
There's more!
• Examples
• Templates
• Tips
• Standards
• FAR
Questions?
References

- ® Performance-Based Earned Value is registered by Paul Solomon in the U.S. Patent and Trademark Office. PBEV is a Service Mark of Paul Solomon.
- ® PMBOK is registered by the Project Management Institute in the U.S. Patent and Trademark Office

© 2006 Paul J. Solomon
References

Acronyms

- BCWP: Budgeted Cost for Work Performed
- BCWS: Budgeted Cost for Work Scheduled
- EV: Earned Value = BCWP
- EVMS: Earned Value Management System
- IBR: Integrated Baseline Review
- IMP: Integrated Master Plan
- IMS: Integrated Master Schedule
- PBEV: Performance-Based Earned Value
- SEP: Systems Engineering Plan
- SOW: Statement of Work
- TPM: Technical Performance Measure
- WBS: Work Breakdown Structure