AN EFFECTIVE UML PROCESS

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

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

PRIOR EXPERIENCE
1954 - 1964 U.S. Marines
1964 - 1965 General Precision, Librascope Div
  Customer Training Instructor, SUBROC and ASROC ASW Systems
1965 - 1982 Teledyne Ryan Aeronautical
  Field Engineer, AQM-34 Series Special Purpose Aircraft
  Project Engineer, System Engineer, Unmanned Aircraft Systems
1982 - 1984 General Dynamics Convair Division
  System Engineer, Cruise Missile, Advanced Cruise Missile
1984 - 1993 General Dynamics Space Systems Division
  Functional Engineering Manager, Systems Development

FORMAL EDUCATION
BA Math, SDSU
MS Systems Management, USC

INCOSE  First Elected Secretary, Founder, Fellow
AUTHOR  System Requirements Analysis (1993, 2006), System Integration, System Validation and Verification, System Engineering Planning and Enterprise Identity, System Engineering Deployment
There’s a Problem Out There

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

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

FUNCTIONAL FACET

PROBLEM SPACE

BEHAVIORAL FACET

VISION

OBJECT FACET

ANALYST
## 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 DIAGRAM</td>
<td>USE CASES AND ACTIVITY DIAGRAMS</td>
<td>STATE, SEQUENCE, AND COMMUNICATION DIAGRAMS</td>
</tr>
</tbody>
</table>

- UNPRECEDENTED ANALYTICAL ENTRY FACET
How Should I Enter Problem Space?

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

Overview

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

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

RAS IN A DATABASE

Specifications

Interface Requirements

Specialty Engineering Requirements
Use a common structure that includes hardware and software.
Apply a top-down analysis for both that contributes to the identification of entities in the common product entity structure.
A Preferred Modeling Order

• Early object oriented analysis encouraged this pattern.

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

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

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

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

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

• Identify an initial product entity that will be developed as computer software using traditional structured analysis.
• Dynamically analyze the entity using UML.
  – Use cases
  – Sequence diagram
  – Communication diagram
  – Activity diagram
  – State diagram

• In the sequence, communication, and activity diagramming analysis you will have to identify next lower tier software product entities.
• And the process continues to expand and move deeper translating problem space into solution space.
• At the bottom are classes about which code can be written based on requirements derived from the dynamic modeling work.
Surprise!

System Analysis Using TSA

Identify Top Level Software Entities
Continued Software Items Analysis Using UML

Identify Top Level Hardware Entities
Continued Hardware Items Analysis Using TSA

Note that the SW work pattern encouraged exactly parallels that employed in TSA.
The Diagrams of UML 2

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

• For modeling static aspects of the system
  – Object and class diagrams
  – Component diagram
  – Deployment diagram
  – Composite structure diagram (2)
  – Package diagram (2)

(2) = added in UML 2.0
The classifier is the product entity the specification is being written for.

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

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

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

Borrowed from Modern Structured Analysis to provide an organized approach to use case identification.
Hierarchical Modeling Relationships
- The Supporting Dynamic Modeling Artifacts

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

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

Actors derive tangible benefits from the system.

Actors Name

Classifier AX

Extend Relationship

Include Relationship

Generalization Relationship

Use Case Name

Extended Use Case

Included Use Case

Generalization Use Case

Generalization Use Case

Generalization Use Case
Use Case Relationships

• **Extend**
  – Pushes common behavior into other use cases that extent a base use case

• **Include**
  – Pulls common behavior from other use cases that a base use case includes

• **Generalization**
  – A child use case inherits behavior and meaning of the base use case
  – The child use case may add or override the behavior of the base use case
Use Case UX11

Actors derive benefits from the system.

Use Case UX11

Use case 1 of terminator 1 for classifier AX.

Extended Use Case UX111

Extended Use Case UX112

Extended Use Case UX113

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

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

Textual scenario descriptions (use case specifications)
Scenario

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

If you want a copy of a DID email jgrady@ucsd.edu
The Other Dynamic Models

Sequence Diagram UX11321

Activity Diagram UX11323

State Diagram UX11324

Interaction Diagrams

Communication Diagram UX11322
Sequence Diagram UX11321
Emphasizes the time ordering of messages

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

• A message is the specification of a communication among classifiers on a class or object diagram or between the classifier represented by life lines on the sequence diagram or blocks of a communication diagram.

• When a message is passed from one classifier to another some action usually results on its receipt.

• The action may result in a change of state in the classifier on the arrow head.

• Describe the related requirements in terms related to the target classifier.
Sequence Diagram Message Types

• Call
  – Invokes an operation on an object represented by the lifeline
  – An object can send a call to itself resulting in a local invocation

• Return
  – Returns a value to the caller

• Send
  – Sends a signal to an object

• Create
  – Creates an object

• Destroy
  – Destroys an object
The UML Static Entities

- **System/Subsystem**
  - The highest level software entity. There can be many of these entities in a real system composed of hardware and distributed software. A collection of subsystems composed of nodes or simply nodes.

- **Node**
  - Appears on a deployment diagram that exists at run time and is a computational resource, generally having at least some memory and often processing capability. A collection of components.

- **Component**
  - A modular part of the system consisting of other components or directly of classes.

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

- **Object**
  - An instance of a class.
Deployment and Component Diagrams
UML Structural Artifacts in a Product Entity Structure

Top-Down Development Encouraged

Component
  ├── Class
  │    ├── Class
  │    └── Class
  └── Component
      ├── Class
      │    └── Class
      └── Component
          ├── Class
          │    └── Class
          └── Component
              ├── Class
              │    └── Class
              └── Class

Dynamic Analysis

Lower Tier Classifiers Identified
A Flexible Dynamic Modeling Overview

1. Context Diagram
2. Use Case
3. Possible Extended and/or Included Use Cases
4. Scenario Set For Each Use Case
5. Activity Diagram for Each Scenario
6A. Sequence Diagram
6B. Activity Diagram With Swimlanes
7. State Diagram
8. Product Entity Structure
9. Dynamic Analysis

Cycle to Lower Tiers

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

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

- SoS
  - System A1
  - System A2
  - System A3
  - System A4
  - System A5
- Segment A31
- Segment A32
- Segment A33
- Element A321
- Element A322
- Element A323
- Element A324
- Element A325
- Element A326
- Element A327
- Element A328
- Element A331

Requirements Traceability Concerns

Downward Traceability Situation

<table>
<thead>
<tr>
<th>FROM</th>
<th>HW</th>
<th>SW</th>
</tr>
</thead>
<tbody>
<tr>
<td>HW</td>
<td>No Problem</td>
<td>Not Going To Happen</td>
</tr>
<tr>
<td>SW</td>
<td>Problem</td>
<td>No Problem</td>
</tr>
</tbody>
</table>

Software Entities
The System Product Entity Structure

- **Hardware entity**
- **Software entity**

This is the kind of relationship of interest

Level at which a subordinate software entity is identified

Development Orientation
Traceability Across 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
Traceability Evaluation Matrix

Requirements Derived From UML Modeling

POSSIBLE RID EXAMPLE

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

Alternatively, one could rely upon experienced inspection without the organizing influence of the matrix.
Save the UML Models Too!
A Near Term Tool Set Solution

- Enhanced FFBD in Core
- N-Square Interface Analysis
- Specialty Engineering Scoping Matrix
- Traditionally Structured Analysis
- Manually Accomplished Environmental Analysis
- Modern Structured Analysis Using STP
- Manually Accomplished N-Square Analysis
- Vertical Traceability
- UML Accomplished with Rational Products

RAS in Doors

Publish Specification
Tools Integration

- Traditional Structured Analysis
- Manual Methods
- Database Loaders
- UML
- Database System Suite
- Data Base Mgmt
Tools Integration

TRACEABILITY TABLE

DOORS IMPLEMENTED RAS

PERFORMANCE AND INTERFACE REQUIREMENTS

UML APPLICATION

SPECIALTY ENGINEERING REQUIREMENTS

SCOPING MATRIX

ENVIRONMENTAL REQUIREMENTS

THREE LAYER ENVIRONMENTAL MODEL

CORE APPLICATION
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>Interaction Diagrams</td>
</tr>
<tr>
<td>Component Diagram</td>
<td>Communication Diagram</td>
</tr>
<tr>
<td>Object &amp; Class Diagrams</td>
<td>Sequence Diagram</td>
</tr>
<tr>
<td>State Chart</td>
<td>Use Case Diagram</td>
</tr>
<tr>
<td>Architecture Block Diagram</td>
<td>Activity Diagram</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Physical Facet</th>
<th>Behavioral Facet</th>
<th>Functional Facet</th>
</tr>
</thead>
<tbody>
<tr>
<td>State Diagram</td>
<td>Schematic Block Diagram</td>
<td>Timeline Diagram</td>
</tr>
<tr>
<td>Functional Flow Diagram</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

## Traditional Structured Analysis
A Subset of UML?
Modeling Changes In the Near Term

UNIVERSAL MODEL OF THE FUTURE

UML

SysML

Component Diagram
Deployment Diagram
Communication Diagram
Interaction Overview Diagram
Package Diagram

REPLACING TSA

SysML DERIVED FROM UML

Requirements Diagram
Parametric Diagram
Assembly Diagram

Use Case Diagram
Activity Diagram
State Diagram
Sequence Diagram
Object/Class Diagram
Timing Diagram
Composite Diagram

PUSH THESE COMPONENTS TOGETHER MORE TIGHTLY
System Modeling Evolution Timeline

- **1920**
  - DOCUMENT DRIVEN DEVELOPMENT

- **1970**
  - DATABASE DRIVEN DEVELOPMENT

- **1990**
  - MODEL DRIVEN DEVELOPMENT

- **2010**
  - NOW
    - RISE IN THE USE OF STRUCTURED ANALYSIS

- **2030**

05-15-2002 DATA UNSUBSTANTIATED
DATES ARE APPROXIMATE
Over the Hill and Through the Woods to Utopia

Traditional Structured Analysis

Flow Charting

1950s

Modern Structured Analysis

Early OOA

HP

1950s

Traditional Structured Analysis

FFBD

IDEF0

EFFBD

BD

HFA

2010s

Utopia

UML

DoDAD

SysML

© JOG System Engineering, Inc.