Leveraging Software Development Approaches in Systems Engineering

Rick Steiner
Engineering Fellow
Raytheon Integrated Defense Systems
fsteiner@raytheon.com

6 May 2004
Naval Postgraduate School SI4000 Project Seminar
We’re going to talk about:

- Why Software Tools exist, why Systems Engineers should care
- Software vs. SE as a discipline – key differences
- The importance of requirements
  - Different requirement/system development approaches
  - Pros & cons of each, and how they relate to software approaches
- How Use Cases relate to Requirements
  - Hints on how to manage use case development
- How Object Oriented Design relates to Functional Analysis
  - or not!
- What graphical languages can help (UML, SysML)
- The promise of Model Driven Architecture (MDA)
Software Development Crisis

• In the 1980’s, software development underwent a crisis:
  – Software was RAPIDLY proliferating
  – Software was becoming very complex
    • Software on top of Software (OS, Application)
    • Software talking to Software (interfaces)
  – Software development delays were holding up system delivery
  – Software was becoming very expensive to develop and maintain
  – Software development effort was becoming very hard to estimate
  – Software reliability was becoming problematic
  – Existing techniques were proving inadequate to manage the problem

• Reasons:
  – Economics
    • Processing hardware (silicon) got cheap
      – Easy way to add capability
    • Cheaper to modify product through software than hardware
• In the ’90’s, software development changed:
  – New methods
    • Scalability – Structured Analysis – Coad/Yourdon
    • Reuse – Object Oriented Design
  – Model based tools & techniques
    • CASE tools – Excellerator, TeamWork, Software through Pictures
    • Software modeling languages & techniques
      – Unified Modeling Language (UML)
        • Object Modeling Technique (OMT) - Rumbaugh
        • Use Cases - Jacobsen
        • Sequence Diagrams – Booch
      – Specific techniques (ROOM, RUP, 4+1, etc.)
        • Estimating models & tools: COCOMO, SEER, Price-S, etc.
• When appropriately applied, these changes dramatically improved the predictability, productivity, and quality of software development!
  – Software began to play a progressively larger role in the product system.
### Differences between SW and Systems

<table>
<thead>
<tr>
<th>Mission</th>
<th>Systems Engineering</th>
</tr>
</thead>
<tbody>
<tr>
<td>Efficiently</td>
<td>1) Ensure requirements correct</td>
</tr>
<tr>
<td>develop software</td>
<td>2) Ensure system works</td>
</tr>
<tr>
<td>that meets</td>
<td></td>
</tr>
<tr>
<td>requirements</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Product</th>
<th>Systems Engineering</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software ready</td>
<td>1) Specifications</td>
</tr>
<tr>
<td>for integration</td>
<td>2) Integrated, usable system</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Lifecycle</th>
<th>Systems Engineering</th>
</tr>
</thead>
<tbody>
<tr>
<td>Development</td>
<td>1) Concept -&gt; Requirements</td>
</tr>
<tr>
<td>(design, code,</td>
<td>2) Integration -&gt; Acceptance</td>
</tr>
<tr>
<td>test)</td>
<td>3) Disposal</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Focus</th>
<th>Systems Engineering</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source code,</td>
<td>Requirements, tests, reports</td>
</tr>
<tr>
<td>diagrams</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Done when</th>
<th>Systems Engineering</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code compiles</td>
<td>1) Requirements balanced</td>
</tr>
<tr>
<td>error free,</td>
<td>2) System accepted</td>
</tr>
<tr>
<td>unit test complete</td>
<td></td>
</tr>
</tbody>
</table>
• In the ’90’s, system development underwent a crisis:
  – Systems were becoming very complex
    • Systems on top of Systems (SoS)
    • Systems talking to Systems (system level interfaces)
  – Systems Engineering delays were holding up software development
  – Systems were becoming very expensive to develop and maintain
  – Systems development effort was becoming very hard to estimate
  – Systems reliability was becoming problematic
  – Existing techniques were proving inadequate to manage the problem

• Reasons:
  – Demand for increased capability
  – Systems becoming software intensive (embedded processing)
  – Decreased Manning driving increased automation
  – Reliability of manned systems and weapon systems cannot be compromised, in spite of rising complexity
In the ’00’s, system development is changing:
- More rigorous approaches to Requirements
- Use of Models to specify systems
- Adoption of successful software modeling methods
  
  - Model Driven Development
  - Hatley-Pirbhai
  - Object Oriented Techniques
- Adaptation of software modeling languages & techniques to systems engineering
  
  - System Modeling Language (SysML)
- Estimating models & tools: COSYSMO
- Development of new methods
  
  - Systems Architecting
Characteristics of a Good System

Development Approach

- Sort wants from needs
  - Identify and relay imperatives
  - Track and tradeoff everything else
- Validate imperatives
- Manage/control level of abstraction
  - Segregate requirements from design at each level of abstraction
- Keep track of Form vs. Functional imperatives
- Provide a framework for assessing completeness of all requirements & design
- Provide a framework for assessing consistency across all requirements & design
- Provide a framework for verifying product meets the requirements
The *traditional* approach:

- Characterized by textual specifications
- Specifications created and managed as documents
- Specifications provided in a hierarchical tree
- Specifications may be parsed and requirements linked in a database
Document Driven Pros & Cons

Advantages:
• Easy to understand, traditional approach
• Clear, straightforward hierarchy of specifications quickly defines levels of abstraction
• In precedented systems, can rapidly partition requirements development task
• Allows loose coupling between requirements developers
  – Can make rapid progress early in program, compared to other methods

Disadvantages:
• Consistency of requirements hard to assess
  – must read many documents, manually link related requirements
• Large “chunks” of requirements unwieldy
  – latencies associated with specification updates are significant
  – need for reparsing/retracing of requirements after each update
• Product tree needs to be defined in advance
  – not amenable to unprecedented systems
• Requirement definition can outpace analysis & design
  – lower level requirements defined before impact at higher level design is understood
• Focus can easily revert to quantity, rather than quality of requirements
Database Driven Approach

Becoming more commonplace in Systems Engineering:
- characterized by integrated requirements/design databases
  - requirements are records in relational database
  - relations between requirements, attributes of requirements emphasized
- “specifications” are views into database
- requirements hierarchy very flexible

![Database Screenshot]
Database Driven Pros & Cons

Advantages:
- **Difficult to defer rigor**
  - need thorough analysis of requirements up front
  - difficult to “cheat” to save time
- **Benefits of clear linkage**
  - on-demand consistency checking
  - facilitated completeness checking
  - on-demand verification
- **Flexible hierarchy**
  - can easily move requirements to appropriate level of detail
- **Rapid cycle time for updates**
  - on-demand change impact assessment
  - clear ownership control
- **Unambiguous linkage to design tools**

Disadvantages:
- **Difficult to defer rigor**
  - need thorough analysis of requirements up front
  - difficult to “cheat” to save time
- **Slow startup... many decisions need to be made up front**
  - requirements heirarchy, multiple heirarchy - need CLEAR vision of what to do!
  - guidelines for requirements attributes
  - specification scripts
  - linkage to design tools
  - training, training, and relevant training
- **Investment in resources**
  - experienced toolsmith
  - experienced process owner
Model Driven Approaches

Becoming more common in Software development
Rarely implemented at Systems Engineering level - high risk, high payoff
- characterized by integrated model that represents both design and requirements
- “specifications” are views into model
- “requirements hierarchy” doesn’t exist by itself
  - “requirements” are simply characteristics of the model

![Functional Allocation: Activity Diagram]

![Functional Allocation: Assembly Diagram]
Model Driven Pros & Cons

Advantages:

• **Strong enforcement** of rigor
  – need thorough analysis of requirements up front
  – impossible to “cheat” to save time
• Clear, unambiguous system definition
  – clear allocation of function onto form
• Benefits of clear linkage
  – on-demand consistency checking
  – facilitated completeness checking
  – on-demand verification
• Possible to eliminate “shall” altogether
  – “firmness” becomes an attribute of model elements
• Very rapid cycle time for updates
  – on-demand change impact assessment
  – clear ownership control
• Unambiguous linkage to design tools

Disadvantages:

• Impossible to defer rigor
  – impossible to “cheat” to save time
• Slow startup... many decisions need to be made up front
  – syntax and relationship of proposed models must be crystal clear!
  – guidelines for model attributes
  – linkage to design tools
  – training, training, training, experience, and relevant training
• Significant up front investment in resources
  – Very experienced toolsmith
  – Very experienced process owner
• The model can become as complex as the product itself
## Development Approach Scorecard

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Document Driven</th>
<th>Database Driven</th>
<th>Model Driven</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sort wants from needs</td>
<td>“Shall” statements</td>
<td>Attributes, link to CONOPS</td>
<td>Attributes of model elements</td>
</tr>
<tr>
<td>Validate imperatives</td>
<td>Manual only</td>
<td>Link to analyses</td>
<td>Model execution, links to analyses</td>
</tr>
<tr>
<td>Manage/control level of abstraction</td>
<td>Spec tree: specification vs. design description</td>
<td>Hierarchy, requirement tree</td>
<td>Product hierarchy, consistency checks</td>
</tr>
<tr>
<td>Form vs. functional imperatives</td>
<td>Typically poor segregation</td>
<td>Attributes, scripts, filters</td>
<td>Separate form, function, and allocation</td>
</tr>
<tr>
<td>Framework completeness</td>
<td>All top level requirements traced to lower level</td>
<td>Vertical linkage, hierarchy</td>
<td>Vertical linkage</td>
</tr>
<tr>
<td>Framework consistency</td>
<td>Typically poor – some peer to peer requirements tracing</td>
<td>Horizontal linkage</td>
<td>Horizontal linkage</td>
</tr>
<tr>
<td>Framework for meeting the requirements</td>
<td>System Requirements Verification Matrix</td>
<td>Link to verification database</td>
<td>Development and verification scenarios</td>
</tr>
<tr>
<td>Semantics captured</td>
<td>Low</td>
<td>Medium</td>
<td>High</td>
</tr>
<tr>
<td>Design iteration time</td>
<td>Long</td>
<td>Medium</td>
<td>Short</td>
</tr>
</tbody>
</table>

Copyright © 2003 Raytheon Company UNPUBLISHED WORK ALL RIGHTS RESERVED
EIA 632 SE Process IDEFØ w/ Models

- All four activities happen in parallel
- Risk Management & CAIV are integral to process
- Process is applied iteratively at each level of design
Unified Modeling Language (UML)

• UML is maintained by the Object Management Group (OMG)
• The Unified Modeling Language (UML) is
  – a *graphical* language for visualizing, specifying, constructing, and documenting the artifacts of a software-intensive system. (from the OMG UML 1.4 specification, emphasis added)
  – the industry standard for expressing and communicating object-oriented software designs
• Has undergone several revisions
  – 1.0 Original submittal - Never released
  – 1.1 UML Partners final submittal - First approved standard
  – 1.2 Editorial clean-up - Document changes, no technical changes
  – 1.3 Revisions, not enhancements - Clarifications and corrections
  – 1.4 Revisions to UML extensions - Released late 2001
  – 2.0 Major revisions to Behavior and Structure
    • Approval August 2003, release expected soon.
• So what does that mean to the systems engineering community
  – The OMG, in cooperation with INCOSE and ISO are exploring ways to expand the role of UML into the realm of systems engineering
UML 2 Diagram Taxonomy

UML 2 Diagram

Behavior Diagram
- Activity Diagram
- Interaction Diagram
- Interaction Overview Diagram
- Communication Diagram
- Timing Diagram

Use Case Diagram
- Sequence Diagram

Class Diagram

Component Diagram
- Object Diagram

Deployment Diagram
- Package Diagram

Structure Diagram
- State Machine Diagram

Systems Engineering Interest
Structure in UML 2

Definition (Class Diagram)

Structural Hierarchy: Class Diagram

Electronic Processor  Anti-Lock Controller  Electro-Hydraulic Valve

Traction Detector  Brake Modulator

Use (Composite Structure Diagram)

Structural Hierarchy: Composite Structure Diagram

Anti-Lock Controller

:modulator interface

:Traction Detector

:Brake Modulator
Extending UML to Systems Engineering

  - joint INCOSE-OMG initiative chartered in 2001- collaborated with UML2
  - drafted UML for SE RFP, issued by the OMG in March 2003
- Systems Modeling Language (SysML) – http://www.sysml.org
  - SysML Partners organized in May 2003 to respond to RFP
    - Industry - BAE SYSTEMS, Deere & Company, IBM, Lockheed Martin, Motorola, Northrop Grumman, Raytheon, Thales
    - Government - NASA/JPL, NIST, OSD
    - Tool Vendors - Artisan, Gentleware, IBM/Rational, I-Logix, Telelogic, Vitech
    - Liaisons - AP-233, INCOSE, Rosetta, EAST, Ptolemy
  - SysML will customize UML 2.0 to support the specification, analysis, design, verification & validation of complex systems.
  - SysML Draft spec presented to INCOSE in January, OMG in February 04
  - SysML 1.0 spec will be submitted to OMG in August 04, expect release in early ‘05
4 Pillars of SysML

Structure

ABS System: Assembly Diagram
- Anti-Lock Controller
  - :Traction Detector
    - <<allocation>>
    - <<activity>> Detect Loss of Traction
  - :Brake Modulator
    - <<allocation>>
    - <<activity>> Modulate Braking Force
- modulator interface
- tracLoss

ABS Spec: Requirements Diagram
- Vehicle System Specification
  - <<requirement>> R102
    - Id: 102
    - text: System shall ..
    - Criticality: H
- Braking Subsystem Specification
  - <<requirement>> R337
    - Id: 337
    - text: Braking subsystem shall ..
    - criticality: H
- <<trace>>

Behavior

Apply Brakes: Activity Diagram
- :Traction Detector
- :Brake Modulator
- Detect Loss of Traction
- Modulate Braking Force
- <<allocation>>
- <<activity>>

Braking Performance: Parametric Diagram
- <<property>>
  - Tire.friction
  - Braking.friction
  - Vehicle.weight
  - Vehicle.speed
- <<parametricRelation>>
  - Total Force = Sum Forces
  - Force = m*a
  - Vehicle.deceleration
- <<property>>
  - Stopping. distance
  - Integrate

Requirements

Parametrics
Object Oriented Analysis (OOA) & Use Cases

• OOA focuses on SERVICES the system is to provide, rather than functions the system performs
• Use Cases are *textual* descriptions of scenarios
  – They usually follow a standard format or template
  – They address sequences - “happy path” and alternate paths
  – They can include diagrams to show sequences/behavior
  – They can address various levels of detail
  – The relationships between Use Cases can be represented in a diagram
Use Case Pros & Cons

Advantages:

• Help segregate problem from solution
  – Services aren’t functions
• Help focus on most important aspects of system
• Used throughout design process, and into testing
  – Basis for test planning
• Vehicle for dialog with customer
• Vehicle for dialog with software developers
• Can be used in conjunction with requirements database to generate specification
  – This is an extension to OOA

Pitfalls:

• Difficult to estimate in advance
• Incomplete
  – Only relate to functional requirements
  – Not performance or non-functional requirements
• Explosion of Use Cases for complex systems
  – Difficult to manage
  – When are you finished?
• Confusion/overlap with functional analysis
  – Services aren’t functions
Managing Use Cases

System Threads
(concatenation of user scenarios)

User Scenarios
(specific sequences)

Use Cases
(actors & interfaces)

Sys Arch Alt (A)

System Model
(representation of system to be built)

Reference & Test Cases
(specification of essential system behavior)

Copyright © 2003 Raytheon Company UNPUBLISHED WORK ALL RIGHTS RESERVED
Object Oriented Development (OOD)

- OOD focuses on maximizing **cohesion** and minimizing **coupling**
  - Maximizing Cohesion: grouping objects together that tightly interrelate
  - Minimizing Coupling: simplifying interfaces between groups of objects, making them as independent as possible
- This makes objects reusable
  - Aids in the “definition – usage” pattern discussed earlier
  - Isolates the behavior and data of each object from every other object

**Advantages:**
- Reusable objects, each self contained
  - Significantly reduces subsequent development time
- Strong interface management
- Proven value on non-realtime software development

**Pitfalls:**
- Extra bulk, overhead that doesn’t add capability in execution
- Cannot separate Form and Function
  - Not amenable to functional specification
- Data is internalized
  - Not amenable to data engineering
Model Driven Architecture (MDA)

• MDA has been developed & promoted by the OMG
  – See also “Executable UML” – Steve Mellor
• Agreement that existing OOD techniques can be too restrictive
  – Need to model patterns, abstract architecture
  – I see this as a way of segregating form (what) from function (how)
• MDA uses two DIFFERENT modeling levels:
  – Platform Independent Model (PIM)
    • All abstract (non-instantiable) classes, no language dependency
    • Focus on grouping of behavior, data, interfaces
    • I call this “logical architecture”
  – Platform Specific Model
    • Specific languages (Java, C++, etc) and compilers
    • Implementation details
  – One PIM can have many compliant PSMs
System Model & Performance Analysis

Requirements

System Alternative (A)

System Model

Analysis Plan

Analytical Models

- closed form
- discrete event
- network

performance budgets

performance estimates

analysis needs

analysis specification
- purpose
- scope
- criteria

function
form

Needs

Requirements

 closed form
 discrete event
 network

Analytical Models

System Model

System Alternative (A)

function
form

Analysis Plan

Raytheon

Copyright © 2003 Raytheon Company UNPUBLISHED WORK ALL RIGHTS RESERVED
Summary

- Systems Engineering needs help to manage development of today’s complex systems
- Software Engineering has a variety of tools and techniques which have proven successful
- Applying Software Engineering techniques to SE needs to be done with a full understanding of the scope of SE objectives
- While advanced model driven techniques are appropriate for complex, unprecedented, ultra-quality systems, these techniques require
  - Training
  - Tools
  - Startup time
- These advanced techniques aren’t ALWAYS appropriate, especially for highly precededented or legacy systems.
Bibliography

- http://syseng.omg.org (OMG SEDSIG site)
- http://www.sysml.org (SysML Partners site)
- Writing Effective Use Cases, A. Cockburn, Addison-Wesley, 2000, ISBN 0201702258
- “Topics in Modern Requirements Development”, R. Steiner and J.M. Green, San Diego INCOSE tutorial
- “System “Late Binding” of Function to Form using UML”, R. Steiner, San Diego INCOSE 2003 mini-conference
- ““Shoot the Modelers & Begin Design”; Focusing Analysis on Design Using a System Model”, R. Steiner, Proceedings of INCOSE Symposium 2001