SYSTEMS ARCHITECTURE
PROCESSES, DECISIONS, ARTIFACTS

Richard Taylor
March 31, 2012
The Systems Engineering “Vee”

1. Understand User Requirements
   - Develop System Specs & System Validation Plan
     - Expand Functional & Design Views to CIs
       - Design-to Specs Build-to Docs
         - Fab, Assemble Code
           - Inspect to Build To Docs
             - Assemble CI And Verify
               - Integrate System And Verify
                 - Demonstrate And Validate System

2. Develop System Specs & System Validation Plan
   - Understand User Requirements

3. Integrate System And Verify
   - Assemble CI And Verify

4. Demonstrate And Validate System
   - Integrate System And Verify

5. Fab, Assemble Code
   - Inspect to Build To Docs

6. Design-to Specs Build-to Docs
   - Expand Functional & Design Views to CIs

7. Understand User Requirements
   - Develop System Specs & System Validation Plan
System Life Cycle Technical Processes

- Stakeholder Requirements Definition
- Requirements Analysis
- Architectural Design
- Implementation
- Integration
- Verification
- Transition
- Validation
- Operation
- Maintenance
- Disposal
Systems Architecture Activities

ORD

Stakeholder Requirements Definition

Requirements Analysis

Architectural Design

Mission Scenario

System Boundary

EICD

Operational Scenario

DFD/CFD

FFBD

Modes/States

ERA - Database

EICD

System Block Diagram
- Generic Physical
- Instantiated Physical
- Operational

A-Spec

MOEs

ERA - enterprise

A-Spec

B-Spec

KPPs/MOPs

Architectural Drivers

EICD

C-Specs

TPMs

Performance Models
Brief overview of the 3 architectural processes

• Stakeholder Requirements Definition
  – What does the customer/client really want?
  – How will the consumer use the system?
  – How will the system evolve and/or age?

• Requirements Analysis
  – What tasks or functions need to be accomplished to give the customer what he really wants?
  – What information or data needs to flow?
  – What control actions need to take place?

• Architectural Design
  – What configuration items will I use?
  – How do these map to the functions?
Architecture as Decision Making

- Joint exploration of requirements and design
- Discover and resolve inconsistencies of requirements and goals
- “Architecting has been accomplished when the fundamental structural decisions about a system have been made, regardless of what sort of architecture document has been produced.”
Stakeholder
Requirements
Definition
Stakeholder Requirements Definition Process

**Controls**
- Applicable Laws and Regulations
- Industry Standards
- Agreements
- Project Procedures and Standards
- Project Directives

**Inputs**
- Source Documents
- Stakeholders’ needs
- Project Constraints

**Activities**
- Elicit Stakeholder Requirements
- Define Stakeholder Requirements
- Analyze and Maintain Stakeholder Requirements

**Outputs**
- Concept Documents
- Stakeholder Requirements
- Measures of Effectiveness Needs
- Measures of Effectiveness Data
- Validation Criteria
- Initial RVTM
- Stakeholder requirements Traceability

**Enablers**
- Organization/Enterprise Policies, Procedures, and Standards
- Organization/Enterprise Infrastructure
- Project Infrastructure
Stakeholder Requirements Definition

Decisions

Activities

- Elicit Stakeholder Requirements
- Define Stakeholder Requirements
- Analyze and Maintain Stakeholder Requirements

Requirements are an **OUTPUT** of the Systems architecture process

- What is the **scope** of work of the system?
- How will the consumer use the system?
- How will the system evolve and/or age?
- What are the **lifecycle effects**?
- What are the **missions**?
- What do we know about the **interfaces**?
- What doesn’t the **customer/consumer/client/caretaker** know or understand?
- Where are requirements likely to **grow**?
- Where is **overachieving** a virtue?
What does the user really want?

• What is the driving problem requiring solution?
• Why now?
• Why did the customer release a competitive RFP?
• What were the alternatives?
• What are the system constraints?
• What functions/levels of performance will the customer pay extra for?
• Where will future expansion likely be required?

Note: The answers to the above questions are rarely found in the statement of work.
Pretext and Subtext

Pretext
An ostensible or professed purpose

Subtext
Unspoken thoughts and motives
The Users want hard things!

- Heuristic: Extreme requirements, expectations, and predictions should remain under challenge throughout system design, implementation and operation.
- Heuristic: Explore the situation from more than one point of view. A seemingly impossible situation might suddenly become transparently obvious.
- Heuristic: In any resource-limited situation, the true value of a service or product is determined by what one is willing to give up to obtain it.
Sources of Requirements

• Requirements are not “gathered”!
• Requirements are discovered, unearthed, negotiated, and derived.
• Statement of work is only one source.
• Other sources:
  – Customer interviews
  – User interviews
  – Legacy systems
  – Marketing intelligence
  – Management approaches
  – Customer biases
  – Regulations
  – Standards
Stakeholder Requirements Definition

Artifacts

What are the missions?

What is the scope of the work?

System Boundary Diagram

What do you know about the interfaces?

Operational Information Exchange Matrix

What are the lifecycle events?

Operational Scenario Diagram
What are the missions?

- The Mission Scenario describes the purpose of the system within the context of an organization mission.
- It provides a framework to:
  - Discover special user requirements unique to missions
  - Gain insight into the system complexity
- The Mission Scenarios are inputs to the Functional analysis
Context Diagram/Mission Scenario Diagram

• Provide context by graphically portraying the larger problem that the customer is trying to solve.
• Position our system within that context.
• Provide a common mental model for the customer and developers.
• Describe how the system will support the customer in satisfying the higher level objective.
Sample Mission Scenario – Decennial Census

- Central System Administration
- DCC Monitoring
- Software Distribution
- Takeover of DCC System Admin

- Identify Respondents
- Derive T13 Data from Paper Returns
- System Administration

- Mail back Responses
- Return UAAs

- Collect SEQ
- Collect GQ

- Checkin Data
- T13 Data
- T13 Acknowledgments

- Acknowledge Receipt of T13 Data
- Prepare Non-response Followup
- Process T13 Data

- Update Leave Forms
- Mail Out Forms
System Boundary Diagram

• Defines the context/.scope of the work to be provided as part of your product or system.
• Define in terms of data ownership
• Identifies the external adjacent systems
• **Heuristic**: Diagram abstraction should result in $7+2$ elements.
A word about boundary diagram interfaces

- Annotate data exchanged between systems
- Use a different visual standard for the 3 basic types of functional interface
  - Broadcast or automatic – one-way on the source’s schedule
  - Cooperative or protocol-based – two way according to agreed upon set of rules (a protocol)
  - Independent or conversation-based – two way with common language but no scheduling or protocol
What do we know about the interfaces?

• Record in the Information Exchange Matrix
• Place to capture corporate knowledge
• Very important to keep a high level of abstraction
• Identify problematic interfaces
  – Interfaces for which we don’t know what, how, how much or how fast
• Don’t focus on implementation details!
Sample Information Exchange Matrix

<table>
<thead>
<tr>
<th>ID</th>
<th>Source</th>
<th>Destination</th>
<th>Description</th>
<th>Format</th>
<th>Protocol</th>
<th>Volume</th>
<th>Timing</th>
<th>Security</th>
<th>Avail</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>System</td>
<td>Sub-system</td>
<td>System</td>
<td>Sub-system</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Write down everything you know at this level.
- Keep the level of abstraction appropriate
- **Heuristic**: It doesn’t exist if it isn’t written down!
What are the operation scenario events?

- The operational scenario diagram is a framework to:
  - Understand how the system is used and operated over its entire life cycle
  - Identify the applicable environments
  - Identify and capture additional life cycle requirements.
Operational Scenario Overview

Lifecycle Path

Feedback Path

Lifecycle Event

Build, Test & Ship to Customer

Storage Prior To Installation

Training Simulation

Installation And Test

Post-Mission Ops

Mission

Pre-Mission Ops

Product Updates & Upgrades

Replacement Retirement

24/3/2012

INCOSE Systems Architecture Seminar
What You Should Know At This Point

• The customer’s motivation for the system
  – Problem to be solved
  – Urgency
• High-level requirements
• Environment
  – Legacy systems
  – Interfacing systems
• How the system will be used
  – Context
  – Users
Requirements Analysis
Requirements Analysis Process

**Controls**
- Applicable Laws and Regulations
- Industry Standards
- Agreements
- Project Procedures and Standards
- Project Directives

**Inputs**
- Concept Documents
- Stakeholder Requirements
- Initial RVTM
- Stakeholder requirements Traceability

**Activities**
- Define the System Requirements
- Analyze and Maintain the System Requirements

**Enablers**
- Organization/Enterprise Policies, Procedures, and Standards
- Organization/Enterprise Infrastructure
- Project Infrastructure

**Outputs**
- System Requirements
- MOP Needs
- MOP Data
- System Functions
- System Functional Interfaces
- Verification Criteria
- Specification Tree
- System Specification
- Updated RVTM
- System Requirements Traceability
Requirements Analysis Decisions

Activities
• Define the System Requirements
• Analyze and Maintain the System Requirements

• Define what the system must do to accomplish the work.
• Translate operational requirements into a functional model.
• Output functional requirements that constrain the behavior of the model.
• These will be the system requirements
• Defer considerations of time and space; focus on dependency between functions rather than performance
System Requirements

• Originating
  – Based on operational needs (mission scenario)
  – Leave substantial room for design flexibility
  – Described in language readily comprehensible by the user or stakeholder.

• Derived
  – Derived from originating requirements
  – Addition details or quantification
  – Sufficient for design teams to do their work

• Implied
  – Not specifically identified in the operational needs
  – Inferred from operational needs (operational scenario)

• Emergent
  – Not identified in the operational needs
  – Stakeholders identify during SE process
Requirements Classification

- **Functional requirements**
  - **WHAT** is required to be done
  - E.g., design a ground vehicle to transport troops

- **Non-functional requirements**
  - **HOW WELL**, (how accurately, how fast, how reliably, etc) a functional requirement is to be performed
  - E.g., carry at least 50 personnel at a speed of at least 50 MPH
  - May quantify the performance of:
    - One specific functional requirement
    - The cooperative behavior of many related functions

- **Constraint**
  - **LIMITATIONS** imposed on implementation
  - E.g., Vehicle must weigh no more than 5 tons
Functional Requirements

WHAT is required to be done

• Dynamic behavior: reaction to events; stimulus/response
• Logical behavior: algorithms
• Temporal behavior: time-based
• Parallelism & concurrency
• External system interactions
• Information relationships
• Flow of control: process dependencies
• Flow of data: data dependencies
• Flow of material
• Persistence of information history
Non-Functional or Quality Requirements

HOW WELL a functional requirement is to be done

- Scalability
- Reliability/MTTR
- Availability/MTBF
- Maintainability
- Survivability
- Durability
- Testability
- Supportability
- Produceability
- Performance
- Volume
- Capacity
- Frequency
- Latency
- Accuracy
- Precision
- Integrity
- Security
- Safety
- Privacy
- Human Factors
- Legal Issues
- Design to cost
- Disposal

These add little extra functionality but define “fitness for a purpose”.
Constraints

LIMITATIONS imposed on implementation

• Weight
• Power
• Size
• Temperature
• Noise

• Vibration
• EMI
• Humidity
• COTS component
• Standards
• Tools
Why Requirements Classification Matters

• Systems architecture (partitioning) is primarily dependent on:
  – Summary functional requirements
    • Potential subsystems
    • Functional requirements can often be refined concurrently with systems architecture development
  – Non-functional requirements that span multiple functions
    • E.g., Availability, Scalability, Throughput
    • May be negotiable; subject to tradeoffs
  – Constraints
    • Usually less negotiable
Levels of Detail for Requirements Specifications

• Functional requirements:
  – Formal, but often at summary level (e.g., “capability”)

• Interface requirements
  – External systems: Formal, detailed!!
  – Users: Informal; evolve from prototyping

• Performance & non-functional: Formal
  – Minimum
  – Desired
  – Maximum growth potential

• Constraints: Formal, detailed
All Architecture is Partitioning

• **Systems architecture** is the partitioning of a system into components of the following types:
  – Software
  – Hardware
  – Communications
  – People
  – Procedures
  – Data

• **Architectural drivers** are those requirements that influence system partitioning
Architectural Drivers

• Use as a checklist for what must be known before proceeding with system architecture
  – Most of the critical partitioning decisions can be made while many of the functional requirements are still being specified.
  – However, it is a waste of time to proceed with system partitioning before resolving all architectural drivers.
Architectural Drivers

- Set system boundary
- Specify external interfaces
- Define major system functions
- Define major system data sets
- Indicate system topology
  - Nature of the requirement
  - Potential solution
- Indicate required system performance
- Specify system environment

Potential subsystems
Set System Boundary

• Systems are **not** 1:1 with projects
  – System boundaries are technically-based
  – Projects are contractually & organizationally-based

• Identify requirements/statements that provide insight into what our system must do vs., what external systems are responsible for.

• Often related to external interfaces
Specify External Interfaces

• Source/sink for data & control
• Watch for:
  – Volume & frequency distribution
  – Data format
  – Physical interface
  – Availability/scheduling requirements
• Anticipate external interface problems
  – Avoid real-time synchronous interfaces, where possible
  – Never use databases as external interfaces
  – Try for low volume, asynchronous, transaction interfaces
External Interfaces

Preferred Method

Our System
- Application
- External I/F
- Data Requests
- Results
- Data Services
- DB

External System
- Data Services
- DB

Alternative Method

Our System
- Application
- Proxy I/F
- Data Requests
- Results

Proxy for External System
- Data Services

External System
- DB
Define Major System Functions & Data

• Look for major groups of related functions
• Look for core retained data – think of a system (and subsystems) as representations (recorded in data) of some part of the real world
• The objective is to include requirements that will suggest potential subsystems.
• Avoid low-level functions.
• Ideally, subsystems will be defined in terms of the state data that they will encapsulate and will include all functions related to that data.
Indicate System Topology

• The nature of the requirement includes physical separation:
  – Where users are
  – Where existing processing centers are
  – Where external interfacing systems are
  – Where sensors must be
  – All spatial constraints

• Physical separation is a potential solution:
  – Non-interfering processes
  – Security isolation
  – Safety isolation
Indicate Required System Performance

• Capacity (how much volume over what time period)
• Scalability (minimum/maximum)
• Accuracy (data content, data timeliness)
• Availability (how is it measured?)
• Response time (end-users, external interfaces)
• End-to-end latency (minimum processing time for each unit of work)

Focus on requirements that cannot be allocated to a single system component
Specify System Environment

• Watch for environmental conditions that will significantly restrict component selection:
  – Deep space
  – Under water
  – Desert
  – Dust
  – Temperature range
  – Electrical interference
  – Humidity range
  – Shock
Architectural Constraints

• Restrictions on implementation
  • COTS/GOTS/NDI
  • Form and fit of the function

• Restrictions on process/methodology
  • design, development, integration,
  • test, delivery, sell-off
  • language
  • standards
  • tools
Requirements Management

• Tag architecturally driving and constraining requirements in requirements database.
• When in doubt, tag them on first pass.
• Remove tags when subsequent analysis indicates:
  – They will not influence partitioning.
  – They can be allocated to a single component within the system.
• Usually 5% to 10% of requirements will be significant drivers or constraints.
• There is only one requirements database – this is a filter to assist the system architect.
Functional and Physical Architectures

1. ORD
2. A-Spec
3. System
4. Site
5. CI
6. B-Spec
7. Component
8. C-Spec

Levels:
- Functional Architecture
- Physical Architecture
Relationship of Functional Requirements/Functions/Components

- **Requirement**: Process that needs to be accomplished
  - Mission Scenario
  - ORD
  - A-Spec

- **Step**: Element within the process necessary to accomplish a requirement
  - 1:1

- **Function**: Normalized step necessary to accomplish 1 or more functional steps
  - 1:* 1:1
  - DFD
  - A-Spec
  - Modes/States
  - B-Spec

- **Component**: Implementation of 1 or more functions
  - 1:* 1:1
  - System Block Diagram
  - C-Spec

- Refine requirements into process steps
- Normalize steps common to multiple processes
- Refine functions to ensure mapping to a unique component
- Functions can support multiple requirements
- Components can support multiple functions

Decompose until cardinality rules are satisfied
Requirements Analysis Artifacts

- DFD/CFD
- FFBD
- Modes/States
- ERA - Database
- EICD
- A-Spec
- B-Spec
- KPPs/MOPs
- Architectural Drivers

Requirements Analysis
Key requirements analysis artifacts

- Function List
- Data Flow Diagram
- Mode/State/Functionality Map(s)
- Measures of Performance & Measures of Effectiveness
- Early Requirements Database
## States/Modes/Functions Defined

### States
- Can be transient
- Frequently user defined
- Rarely user controlled
- Temporally unique (one at a time)

### Modes
- Tend to be user controlled
- Tend to be design defined
- Tend to be “permanent”
- Should be temporally unique

### Functions
- \( Y = F(X) \)
- Transformation of inputs to outputs
- Can have input parameters (control signals) and variables

### Example
- Possible states for a GPS system
  - OFF
  - SELF TEST
  - ACQUISITION
  - TRACK

- Possible Modes for a GPS system
  - Search
  - Position
  - Normal Navigation
  - Precision Navigation
  - High Integrity Navigation

- The GPS function transforms RF inputs from the satellites and command inputs to provide position and/or navigation outputs.
# Mode/State/Functionality Map

## Purpose
- Identify the relationship between modes, states and functions.
- Helps to communicate with the customer.
- Helps to identify incomplete relationships.

## Prerequisites
- Function list
- Mode definition table
- State definition table

## Common Failure Modes
- **DO NOT** dive into implementation!

### Functional Capabilities

<table>
<thead>
<tr>
<th>Modes</th>
<th>States</th>
<th>Functional Capabilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal Operational Mode</td>
<td>Fully Operational</td>
<td>Perform Multi-Sensor Selection</td>
</tr>
<tr>
<td></td>
<td>Initialization System</td>
<td>Handle Crew Controlled Nav Options</td>
</tr>
<tr>
<td></td>
<td>degradation</td>
<td>Perform RNP-Based Navigation</td>
</tr>
<tr>
<td></td>
<td>Fully Operational</td>
<td>Tune VHF Radios</td>
</tr>
<tr>
<td></td>
<td>Initialization System</td>
<td>Provide Navaid Data</td>
</tr>
<tr>
<td>Embedded Training Mode</td>
<td>Fully Operational</td>
<td>Control Navigation Mode</td>
</tr>
<tr>
<td></td>
<td>Initialization System</td>
<td>Simulate airborne environment</td>
</tr>
<tr>
<td>Maintenance Mode</td>
<td>Fully Operational</td>
<td>Perform Built In Test</td>
</tr>
<tr>
<td></td>
<td>Initialization System</td>
<td>Update Embedded Software</td>
</tr>
</tbody>
</table>

3/31/2012

INCOSE Systems Architecture Seminar
Functional data flow & control flow diagrams

• Data flow diagrams and control flow diagrams are the step below System Boundary diagrams
• Graphical depiction of interactions between functions
• Stay focused on the functions, not the implementation
• Functions
  – Verb phrases, e.g., “Compute path deviations”
• Show data and control flows differently
  – Hatley-Pirbhai use solid & dashed
  – IDEF0 uses entry/exit point
• Define the interfaces in an Interface Table or Data Dictionary
Developing the detailed functional architecture

• Composition vs. Decomposition
  – Bottom-up vs. Top Down

• Composition
  – Identify simple functionalities associated with simple scenarios involving a single output of the system
  – Low-level functions (3rd or 4th level) in the functional hierarchy
  – Group low level functions into logical functional blocks
  – Specify interconnects

• Decomposition
  – Partitions top-level system functions, conserving all inputs and outputs.
  – Iteratively continue partitioning to an appropriate level, creating simpler and simpler functions.
  – Depends on sound higher-level definition!
Defining functions

- Hatley-Pirbhai 6-block functional architecture of a generic system or high level function
- The 6-block architecture is fractal: decomposition of each block is another 6-block model
6 block model continued

- Provide user interface
  - Requesting/obtaining user input
  - Provide user feedback
  - Provide outputs
  - Respond to queries

- Format inputs
  - Receive inputs from sources external to function (non-human)
  - Process inputs into a form usable by the function

- Process Model
  - Transformation of the inputs to the outputs

- Control Process
  - Control the activation or order of subprocesses

- Format outputs
  - Process output to form usable by external interfaces
  - Provide output to the interfaces

- Maintenance, etc.
  - Support the primary functionality of the system or function
Measures of Performance & Effectiveness

• Measure of Effectiveness – Customer view
  – The metrics by which a customer will measure satisfaction with products produced by the technical effort.
  – Output of the operational analysis tasks.

• Measures of Performance – Developer view
  – An engineering performance measure that provides design requirements that are necessary to satisfy a measure of effectiveness.
  – There are generally several measures of performance for each measure of effectiveness.
  – Intermediate metrics that provide early indications of success or failure.
What You Should Know At This Point

• The functional steps necessary to perform the required operations
• Developmental and environmental constraints
• Architectural drivers
• Functions required in various modes and states
• Criteria for success
Architectural Design
Architectural Design Process

**Controls**
- Applicable Laws and Regulations
- Industry Standards
- Agreements
- Project Procedures and Standards
- Project Directives

**Inputs**
- Concept Documents
- System Requirements
- System Functions
- System Functional Interfaces
- Specification Tree
- System Specification
- Updated RVTM
- System Requirements Traceability
- Life Cycle Constraints

**Activities**
- Define the Architecture
- Analyze and Evaluate the Architecture
- Document and Maintain the Architecture

**Enablers**
- Organization/Enterprise Policies, Procedures, and Standards
- Organization/Enterprise Infrastructure
- Project Infrastructure

**Outputs**
- Technical Performance Measure (TPM) Needs
- TPM Data
- System Architecture
- Interface Requirements
- System Element Requirements
- System Element Descriptions
- System Element requirements Traceability

3/31/2012
Architectural Design Decisions

**Activities**
- Define the Architecture
- Analyze and Evaluate the Architecture
- Document and Maintain the Architecture
- Create the physical model of the system
- Specify how the system implements the requirements
- Specify hardware and software components
- Define the interfaces that connect components
- Define the behavior of the components
- Allocate requirements to Configuration Items (CIs)
- Factor in time and space considerations
Two Types of Physical Architectures

<table>
<thead>
<tr>
<th>Generic Architecture</th>
<th>Instantiated Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Contains components that support all required functions</td>
<td>• Starts with generic architecture</td>
</tr>
<tr>
<td>• Includes support resources for operation, maintenance, distribution, training, etc.</td>
<td>• Includes complete definitions of performance characteristics</td>
</tr>
<tr>
<td>• Does not specify performance characteristics of each physical resource.</td>
<td>• Allocates specific COTS products or custom development to components</td>
</tr>
<tr>
<td>• Contains components that can be implemented multiple ways.</td>
<td>• Identifies quantities of each component</td>
</tr>
</tbody>
</table>

**Heuristic:** Except for good and sufficient reasons, generic and instantiated structuring should match
The architectural design process

Architectural Design

Create Generic Architecture

Partition into Components
Assign External Interfaces to Components
Assign major functions & data to Components
Record Messages between Components

Create Instantiated Architecture

Identify Instantiated Components
Document Physical Interfaces
Allocate Functional Requirements to Components
Allocate Non-Functional Requirements to Performance Model
Summary of Generic Architecture Development

- Define logical segments (use partitioning guidelines)
- Assign external interfaces to segments (prefer each interface owned by one segment)
- Assign major functions and data sets to segments
- Define internal interfaces between logical segments
- Record messages between:
  - Components
  - Component and internal interface
  - Component and external interface
Significance of System Partitioning

• Partitioning choices are the most important decisions that a system architect makes!
  – System boundary partitions your system from the rest of the universe
  – Internal partitioning:
    • Will either simplify or complicate the development and maintenance process
    • Will either make the system robust or fragile
• It is very difficult to overcome poor partitioning choices
  – Make decisions carefully
  – Make decisions consciously
Partitioning Summary

• Divide system into a set of highly autonomous subsystems
  – Minimize interfaces
  – Encapsulate data
  – Reduce complexity

• Each subsystem should be the responsibility of a single IPT
  – Limit IPT-to-IPT communications to interfaces
  – Each subsystem should be capable of being tested by itself
Designing for Replicability

• The single most important consideration in system partitioning

• The basis for architectural solutions for many architectural drivers:
  – External interfaces – multiple instances
  – System topology – multiple instances of site types, sensors, etc.
  – High availability – reserve capacity
  – Capacity uncertainty, growth, or reduction - scalability
Replication Guidelines

• Isolate functions that do not lend themselves to replicability from those that do.

• Try to make components replicable even if you don’t initially see the need.
  – Many performance requirements are unknowable; virtually all will change
  – Be prepared to accept new technology, e.g., better price/performance

• Beware of components that cannot be replicated - Possible design limit
Replication Guidelines (cont.)

• Make replicable components stateless, if possible.
  – No retained memory between units of work
  – Definition of unit of work is dependent on time horizon

• Otherwise, implement recovery strategy externally from replicable components
  – Minimize where state data must be maintained
  – Bullet-proof state data
Stateless Replicable Components - Example

WORKFLOW MANAGER

Manager
• Distribute work, on request
• Cancel & redistribute uncompleted work
• Integrate completed work

In-baskets

Out-baskets

Stateless Replicable Components

Process Flow

Shared Data Model

Bullet Proof Component

In-baskets

Out-baskets

Manager

Workflow Agent (M)

Process A
• Request work
• Process
• Return completed work

Workflow Agent (N)

Process B
• Request work
• Process
• Return completed work

Workflow Agent (P)

Process X
• Request work
• Process
• Return completed work

In-baskets

Out-baskets

Shared Data Model

Bullet Proof Component

In-baskets

Out-baskets

Manager

Workflow Agent (M)

Process A
• Request work
• Process
• Return completed work

Workflow Agent (N)

Process B
• Request work
• Process
• Return completed work

Workflow Agent (P)

Process X
• Request work
• Process
• Return completed work

3/31/2012 INCOSE Systems Architecture Seminar
Workflow Manager Agents
Power of Replicable Components

• Replicable components can be added/deleted to modify availability.
• Replicable components can be added/deleted to alter capacity.
• New technology can be introduced with minimal impact.
• Performance of full production configuration can be extrapolated from performance of a subset in test lab.
Cascading Replicability

Design replicability into solution at multiple levels

Pool of Data
Introduction Components

Many-To-Many I/F

Replicable Clusters

Many-To-One I/F

Replicable Components

External Interface
Architectural Templates

• Distribution alternatives for components
  – Data
  – Application
  – User interface

• Connectivity alternatives between components

• Process control mechanisms
Distribution Templates

• State vs. replicated/cached/summarized data

• Data distribution
  – Hierarchical
    • Single parent node
    • Multiple parent nodes
  – Federated
    • Partial mesh
    • Full mesh
Hierarchical Data Storage Strategies

<table>
<thead>
<tr>
<th>Central</th>
<th>Regional</th>
<th>Local</th>
</tr>
</thead>
<tbody>
<tr>
<td>State</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>State</td>
<td>Cache</td>
<td>None</td>
</tr>
<tr>
<td>State</td>
<td>Cache</td>
<td>Cache</td>
</tr>
<tr>
<td>Summary</td>
<td>Summary</td>
<td>State</td>
</tr>
<tr>
<td>Summary</td>
<td>State</td>
<td>Cache</td>
</tr>
</tbody>
</table>
Hierarchical Application Strategies

<table>
<thead>
<tr>
<th>Central</th>
<th>Regional</th>
<th>Local</th>
</tr>
</thead>
<tbody>
<tr>
<td>Master</td>
<td>Copy</td>
<td>Copy</td>
</tr>
<tr>
<td>Master</td>
<td>Copy</td>
<td>Thin Client</td>
</tr>
<tr>
<td>Master</td>
<td>Thin Client</td>
<td>Thin Client</td>
</tr>
<tr>
<td>CM Master</td>
<td>Master</td>
<td>Copy</td>
</tr>
<tr>
<td>CM Master</td>
<td>Master</td>
<td>Thin Client</td>
</tr>
<tr>
<td>CM Master</td>
<td>Copy</td>
<td>Master</td>
</tr>
</tbody>
</table>
Hierarchical Connectivity Strategies

- **Regional Node**
  - Parents: Central
- **Local Node**
  - Parents: 1 Regional Parent (n), N Regional Parents (m)
- **Central Node**
  - Parents: Central

3/31/2012 INCOSE Systems Architecture Seminar
Federated Data Storage Strategies

<table>
<thead>
<tr>
<th>Region</th>
<th>Other Region</th>
<th>Local</th>
</tr>
</thead>
<tbody>
<tr>
<td>State</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>State</td>
<td>Cache</td>
<td>None</td>
</tr>
<tr>
<td>State</td>
<td>Cache</td>
<td>Cache</td>
</tr>
<tr>
<td>State</td>
<td>None</td>
<td>Cache</td>
</tr>
<tr>
<td>Copy</td>
<td>Cache Copy</td>
<td>State</td>
</tr>
<tr>
<td>Copy</td>
<td>None</td>
<td>State</td>
</tr>
</tbody>
</table>

Cache can be either copy of data or ID of data
Federated Application Strategies

<table>
<thead>
<tr>
<th>Central</th>
<th>Regional</th>
<th>Local</th>
</tr>
</thead>
<tbody>
<tr>
<td>CM Master</td>
<td>Copy</td>
<td>Copy</td>
</tr>
<tr>
<td>CM Master</td>
<td>Copy</td>
<td>Thin Client</td>
</tr>
<tr>
<td>None</td>
<td>CM Master</td>
<td>Copy</td>
</tr>
<tr>
<td>None</td>
<td>CM Master</td>
<td>Thin Client</td>
</tr>
</tbody>
</table>

Central CM

Regional Node

Local Node

3/31/2012 INCOSE Systems Architecture Seminar
Federated Connectivity Strategies

<table>
<thead>
<tr>
<th>Regional Connectivity</th>
<th>Local Connectivity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full mesh</td>
<td>Single parent</td>
</tr>
<tr>
<td>Full mesh</td>
<td>Multiple parents</td>
</tr>
<tr>
<td>Partial mesh</td>
<td>Single parent</td>
</tr>
<tr>
<td>Partial mesh</td>
<td>Multiple parents</td>
</tr>
</tbody>
</table>
Workflow Processing

Enterprise Work Manager

- Create WF Instances
- Assign Workgroups
- Forward WF Instances
- Maintain worklists
- Provide step data to service providers
- Stage local work
- Decompress images
- Execute applications
- Notify server of completion

WF Templates

<table>
<thead>
<tr>
<th>TX Code</th>
<th>WF Template</th>
</tr>
</thead>
<tbody>
<tr>
<td>TX 1</td>
<td>S1 S2 S3</td>
</tr>
<tr>
<td>....</td>
<td>....</td>
</tr>
<tr>
<td>TX n</td>
<td>S6 S7 S8 S9</td>
</tr>
</tbody>
</table>

TX 1 Submission Data

WF Instance K

WF Instance Data

<table>
<thead>
<tr>
<th>WF Instance ID</th>
<th>Submission &amp; generated data</th>
<th>WF state</th>
</tr>
</thead>
<tbody>
<tr>
<td>K</td>
<td>Instance K data</td>
<td>S1 S2 S3 S4</td>
</tr>
<tr>
<td>Z</td>
<td>Instance Z data</td>
<td>S6 S7 S8 S9</td>
</tr>
<tr>
<td>....</td>
<td>.....</td>
<td>....</td>
</tr>
</tbody>
</table>

Workgroup Server

- Maintain worklists
- Provide step data to service providers

SP 1 Work Station

Local Work Queue

<table>
<thead>
<tr>
<th>Step</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>Instance K Data</td>
</tr>
<tr>
<td>....</td>
<td>....</td>
</tr>
<tr>
<td>S7</td>
<td>Instance Z Data</td>
</tr>
</tbody>
</table>
Publish/Subscribe Processing Models

• Data selection
  – Topic-based
    • List-based
    • Broadcast-based
  – Content-based
• Subscription type
  – Fixed
  – Dynamic
• Push/pull
  – Publisher push
  – Subscriber pull
List-Based Publisher/Subscriber Model

Monitored DB

Publisher

Message

Distributor

Subscriptions
-ID
-Topic

Subscriber 1

Message

Subscriber n

Publisher

Subscriber
Broadcast-Based Publisher/Subscriber Model

- **Publisher**: Monitored DB
- **Message Distributor**: IDs of Subscription Servers
- **Filter 1**: Message
- **Filter n**: Message
- **Subscriber 1**: Subscriptions -ID -Topic
- **Subscriber n**: Subscriptions -ID -Topic

3/31/2012 INCOSE Systems Architecture Seminar
Internal & External ICDs

• Purpose
  – Define the interfaces between systems, sub-systems, components, etc.
  – Good definition and control of interfaces, at all levels, is key to developing robust, well designed systems.
  – Enables modularity and development consistency across large systems.

• Use artifacts generated from models developed earlier in the process
  – They provide the framework for the necessary ICDs
  – Consistent use of artifacts prevents or highlights definition errors before they turn into implementation errors

• Consider broadcast data and cooperative data
• Preferentially consider layered model architectures
• Preferentially consider network-centric solutions
How Interfaces are Documented

• External Interfaces
  – Formal EICD (External Interface Control Document)
  – Contract between systems reflecting
    • Data content
    • Data format
    • Volume
    • Frequency
    • Performance
    • Availability
    • Security
    • Delivery mechanism
  – Under contract level configuration control (i.e., customer approval required to change)

• Internal Interfaces
  – Usually a separate section within subsystem design documents
  – Similar content to external descriptions
  – Under internal configuration control
EICD Outline

• Purpose
• Interface diagram
• Interface Name
  – Connecting systems or components
  – Data description (range of valid values, data types)
  – Data format (XML, comma delimited)
  – Communication protocol/ mechanism (FTP, Oracle AQ)
    • Acknowledgements
    • Exception processing
  – Frequency distribution (e.g., bursty, steady state)
  – Volume
  – Security (e.g., encryption)
  – Performance
    • Response latency
    • Availability service level agreement
Do Users Require External Interface Definition?

• Yes, but users may be external or internal

• External:
  – User performs additional functions that are not integrated into our system
  – View external user as a system of its own
  – Formal EICD is required, although usually called a user guide

• Internal:
  – User functions are limited to interacting with our system
  – View internal user as a component of our system
  – Document interface within design documents, and in user guides

• All user interfaces should be documented in the Operational Concept Document, but not usually at the same level as in the ICD
What You Should Know At This Point

• The components that must be built or acquired to implement the system
• Interface commitments
• How the architecture can accommodate changes in:
  – Function
  – Capacity
  – Performance
• Development can now proceed
Contents of Operational Architecture

• From Instantiated Architecture:
  – Functional architecture allocated to physical architecture
  – Derived interfaces
  – Technology & COTS components
  – Qualification requirements for all components
  – Interface architecture

• New Features:
  – Quantification of replicable components
    • Availability
    • Capacity
    • Geographic distribution
  – Integration with operational environment
Features of Operational Architecture

• Maps to actual operational environment
  – Real locations
  – Real operators
  – Real quantities

• Capable of operational modeling
  – Number of operators of all types
  – Shift considerations
  – Maintenance considerations
    • Sparing approach
    • Service level agreements
  – Facilities requirements
    • Floor space
    • Power
    • A/C
Objectives of Operational Architecture

- Maximize the Fundamental Objective – customer mission
  - Avoid shared responsibilities, if possible
  - Users and automated functions should be as autonomous as possible
- Mitigate Operational Risk – support early critical testing
- Simplify interfaces – Internal & External
  - To maintain availability, avoid cascading dependencies
  - Look for natural partitioning points
    - Low volume
    - Simple data structure
    - Asynchronous dependency
  - Minimize context awareness
Operational Architecture Artifacts

• Usually documented by extending the physical architecture artifacts
  – System Block Diagram
  – Component Allocation Diagram
• Bill of Material (BOM)
Summary

• Generation of an responsive/robust systems architecture is:
  – Difficult
  – Absolutely critical to success

• Three distinct phases:
  – Stakeholder requirements definition
  – Requirements analysis
  – Architectural design
QUESTIONS?