Product Family Engineering

Enabling Shared Use of Product Definition
Why Focus on Product Family Engineering?

• “Recycling” is costly
  - Recycle = copy and paste for EVERY project
  - Commonality of a product is lost across multiple projects - divergence
  - Projects collectively lose cost savings from true reuse
  - Product family variability difficult to manage

• Current product definition difficult to impossible to reuse
  - Requirements were not written with tailoring or reuse in mind
  - Relationships between requirements and architecture misunderstood
  - Requirements express specific design
  - Product definition seldom used 100% - nearly always a deviation to create a new variant – otherwise there wouldn’t be a new ‘project’!
What does it mean to “Reuse”?  

- **Forms of Reuse**
  - **Copy/Paste aka Clone and Own aka Salvage Engineering**
    - “Wild West” Product Development
    - Pro
      - Highly adaptable to needs of the customer
      - Mix and Match as needs dictate
      - Lower overall complexity to a project’s development effort
  - Con
    - Little shared commonality – new features invisible to other projects
    - Product Divergence – no “product” identity
    - Reinvention of the wheel – lack of awareness
    - When “fix bugs”, must fix them everywhere they’ve been copied
      - And no assurance that you know “everywhere”
What does it mean to “Reuse”? 

- Forms of Reuse 
  - Product Reuse Shelf 
    - “Big Brother” Product Development 
    - Pro 
      - Maximum commonality of product identity 
  
  - Con 
    - Project teams discouraged from making changes 
    - Product may not be wholly suitable for needs of customer as is 
    - Tendency to over design a solution – “golden solution” 
    - Tendency to make the simple overly complex 

- Assumes product artifacts as configured are suitable for all potential consumers 
- Change introduces “yet another variant” 
  - How do you manage a multitude of variants? 
  - Reuse Shelf becomes cluttered with variants – is this really better than clone and own?
What does it mean to “Reuse”? 

- Forms of Reuse
  - Run-Time Variant Configuration
    - Add straps to product to dynamically reconfigure product behavior or characteristics
    - “Object Code Reuse”
    - Pro
      - Highly configurable for customer needs
    - Con
      - Infinite number of variations to be managed?
      - Highly complex configuration code to make the product adaptable
      - Customer/ project team don’t really know what they are buying
        » Definition lacks specific configuration definition for “my customer”

- How is common source data (reqts, design, interfaces, test, code..) managed for a common product?
- Works well for software – what about systems and hardware variant management?
What does it mean to “Reuse”?  

- Forms of Reuse  
  - Compile-Time Variant Configuration  
    - Compiler switches used to tailor code in/out at time of compilation  
    - “Source Code Reuse”  
    - Pro  
      - Tailor product for individual customer needs  
      - Simplify configuration control code  
    - Con  
      - Switches difficult to manage when many combinations/configurations are possible  
      - Customer/project team don’t really know what they are buying  
        » Definition lacks specific configuration definition for “my customer”  

- How is common source data (reqts, design, interfaces, test, code..) managed for a common product?  
- Works well for software, but what about systems and hardware?
What does it mean to “Reuse”? 

- **Forms of Reuse**
  - Development Time Variant Configuration aka Product Family Engineering
    - Product definition shared between projects – requirements, architecture, design, interfaces, test, etc
    - “Product Information Shared Use”
  - Pro
    - Common information objects shared across projects
    - Project unique perspectives – see only “my project” definition within context of holistic product definition
  - Con
    - Object level (a requirement, an interface, a component...) configuration management necessary
Project Engineering vs. Product Family Engineering

- **Project Engineering**
  - Devoted to satisfying an end customer
  - Typically oriented toward one customer’s needs
  - Team will do anything for the good of the customer as long as it’s within the current budget and scope.
  - Salvaging
    - Salvage engineering is a key characteristic of a project oriented business
    - Copy and paste
    - Tendency to reinvent
    - Focus on unique customer needs with little or no unified product vision

- **Product Family Engineering**
  - Devoted to satisfying a market need expressed by multiple end customers
  - Team considers alternative strategies to satisfy as many customers as possible, yet stay within project budget and scope
  - Sharing
    - Projects share common product definition - link to shared product intellectual property
      - Common source data for each project
    - Focus on product definition across diverse customers needs, while also enabling custom tailoring
  - Project definition visible from the context of the product definition
Product Family Engineering

- **History**
  - **1996**: Began as requirements engineering process improvement exercise
    - Investigate reasons requirements definition quality and content was so diverse across several projects that supposedly had a common product basis
  - **1997-1998**: Attempt documentation merger of multiple project Software Requirement Specs (SRS) into one Common SRS
    - Merger completed using standard document editing tools
    - Discovered several patterns and behaviors that needed to be rethought
      - Educate engineers on different approach to requirements writing
    - Created the Common SRS with mixed reviews
      - Software teams liked ease of finding differences between projects
      - Program Management, System Engineering teams and customers disliked lack of single project definition ("What is my project definition?")
    - Need ability to filter SRS for "my project" view of data

Common SRS – Project Sharing: USAF (x4), US Army, Middle East, Australia
Product Family Engineering

• History (contd)
  – 1999-2000: Continue to research tools that could support shared requirements between projects, yet enable custom views
    • SLATE identified as potential candidate – lacks needed capability for configuration management control and sharing between projects
    • DOORS identified as potential candidate – lacks system architecting capability
      – DOORS selected as enterprise tool of choice for requirements engineering separately from the Product Family Engineering research
      – Promised development for extending DOORS capability to address PFE concepts has never materialized
Product Family Engineering

• History (contd)
  – 2001-2002: Migrate product definition into DOORS & continue study of patterns of behavior for effective shared product definition
    • Further tool studies fail to materialize tools supporting PFE concepts
    • System Architecting concept is realized as central feature of a PFE environment
      – All shared data must be managed within context of the system architecture
  – 2002-2004: Proof of concept demonstration
    • Created a shared environment for several Flight 2 System projects within DOORS
    • Partial Success:
      – Original program from DOORS migration refused to share database
        » Lack of configuration management to ensure that original project could not be unintentionally altered by sharing
      – Several successive projects share common requirements
        » Custom views provided for each project
        » CM process based on trust and forgiveness
      – Project split due to reallocated/rearchitected project definition
        » DOORS insufficient to address common requirements that could be shared between projects but functionality resides in different architectural components – proof that system architecture is central feature of PFE capability

Common DOORS Flight 2 Navigation SRD – Project Sharing:
• USAF (original source, but not shared);
• Middle East, US Coast Guard (still sharing - based on USAF);
• USN (x3) (originally shared with Middle East/US Coast Guard, but split due to rearchitecture), Korean, USAF (still sharing with USN);
• Germany (originally shared with Middle East/US Coast Guard, but minimal commonality and foreign national access forced need to separate)
Product Family Engineering

• History (contd)
  - 2005+: Continued informal research and investigation into Product Family Engineering capabilities
  - Continue seeking tools that support foundational principles
    • Object level configuration management
    • Objects managed within context of product architecture (logical and physical)
  - Continue to seek sponsorship to finance necessary tool modifications, best practice definitions, and corresponding training
Current State of Practice

- “Document-Centric Configuration Management”

Requirements managed within context of documents and document paragraphs.
Current State of Practice

- “Document-Centric Configuration Management”

**Navigation**

- When $X$ is True, the active waypoint shall be displayed with accents.
- When $X$ is False, the active waypoint shall be displayed with no accents.
- When $Y$ is less than 0, the future waypoints shall be displayed in sequence.

**Guidance**

- When $K$ is Active, the next event shall occur.

For a single project, configuration management is simple.
Current State of Practice

- “Document-Centric Configuration Management”

**Navigation**

- When \( X \) is True, the **active waypoint** shall be displayed with **positive accents**.
- When \( X \) is False, the **active waypoint** shall be displayed with **negative accents**.
- When \( Y \) is less than 0, the **future waypoints** shall be displayed in sequence.

**Guidance**

- When \( K \) is Active, the **next event** shall occur.

Make necessary changes – publish new version of document.
Current State of Practice

- “Document-Centric Configuration Management”

**Navigation**
- When $X$ is True, the active waypoint shall be displayed with accents.
- When $X$ is False, the active waypoint shall be displayed with no accents.
- When $Y$ is less than 0, the future waypoints shall be displayed in sequence.

**Guidance**
- When $K$ is Active, the next event shall occur.

When using a common “source document”, things get complicated. Starts easy with the first project (business as usual).
Current State of Practice

- “Document-Centric Configuration Management”

### Navigation

- When $X$ is True and $Y$ is FALSE, the **active waypoint** shall be displayed with accents.
- When $X$ is False, the **active waypoint** shall be displayed with no accents.
- When $Y$ is less than 0, the **future waypoints** shall be displayed in sequence.

### Guidance

- When $K$ is Active, the **next event** shall occur.

**False**
- Then a new project joins the fray and adds or modifies content.
- New content only applies to new project – and not to original.

**True**

**Choices:**
1) Separate document?
2) Shared requirements – and original project will accept new requirements in v2.0 of document
3) New requirement – with two different variants?
Current State of Practice

- Option 1 ~ Copy/Paste aka “Clone and Own”

---

**Navigation**

- When \(X\) is True, the **active waypoint** shall be displayed with accents.
- When \(X\) is False, the **active waypoint** shall be displayed with no accents.
- When \(Y\) is less than 0, the **future waypoints** shall be displayed in sequence.

---

**Guidance**

- When \(K\) is Active, the **next event** shall occur.

---

**KC-135 FMS System Reqts**

Results in copies of all requirements be made into separate document.

**C-130 FMS System Reqts**

When \(X\) is True and \(Y\) is FALSE, the **active waypoint** shall be displayed with no accents.

When \(X\) is False, the **active waypoint** shall be displayed with no accents.

When \(Y\) is less than 0, the **future waypoints** shall be displayed in sequence.

When \(K\) is Active, the **next event** shall occur.
Current State of Practice

• Option 2: All projects inherit all updates

Common FMS System Reqs

Kc-135 V2.0  C-130 V1.0

Navigation

- When X is True and Y is FALSE, the active waypoint shall be displayed with accents.
- When X is False, the active waypoint shall be displayed with no accents.
- When Y is less than 0, the future waypoints shall be displayed in sequence.

Guidance

- When K is Active, the next event shall occur.

Assumes changes are broadly applicable to all sharing projects.

..but what if “original” project(s) either:
- Cannot accept the change – because of timing (“We’re one month from TRR!”)
- Cannot accept the change – because it just ain’t so on “my project”.

© Copyright 2010 Rockwell Collins, Inc.
All rights reserved.
Current State of Practice

- Option 3: New requirement variation

**Navigation**

- When \( X \) is True and \( Y \) is FALSE, the **active waypoint** shall be displayed with accents.
- When \( X \) is True, the **active waypoint** shall be displayed with accents.
- When \( X \) is False, the **active waypoint** shall be displayed with no accents.
- When \( Y \) is less than 0, the **future waypoints** shall be displayed in sequence.

**Guidance**

- When \( K \) is Active, the **next event** shall occur.

---

<table>
<thead>
<tr>
<th>True</th>
<th>False</th>
</tr>
</thead>
<tbody>
<tr>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
</tr>
</tbody>
</table>

Solves a short-term problem, but for EVERY change, a new requirement must be introduced.

Each project creates a filtered view of “my project”.

What if the requirement is an update that should be globally applied to all sharing projects?
Current State of Practice

• “Document-Centric Configuration Management”
  – DOORS “successfully” used in a shared environment
  – However:
    • Difficult for project team member trust – must be disciplined to copy/paste individual requirements BEFORE modification
    • DOORS baselining control at module level
      – When successive projects “baseline” the module for their next release, all Rev history “erased” – no change bars or change highlighting
      – DOORS “compare” function between baselines ineffective and not “documentable” (does not export to a document format)
    • When faced with sharing project that intends to rearchitect a product (re-allocate functions to new components) causes sharing breakdown
      – Objects can no longer be shared within same “document”, since documents are written according to component scope
      – Reallocation moves requirements to new document!

..but what if we rethink configuration management of documents…. 
Future State of the Practice

• “Information/Model Centric Configuration Management”

Common FMS System Reqts

<table>
<thead>
<tr>
<th>KC-135 V1.0</th>
<th>C-130 V1.0</th>
</tr>
</thead>
</table>

**Navigation**

R34

When X is True and Y is FALSE, the **active waypoint** shall be displayed with accents.

R34

When X is True, the **active waypoint** shall be displayed with accents.

R35

When X is False, the **active waypoint** shall be displayed with no accents.

R36

When Y is less than 0, the **future waypoints** shall be displayed in sequence.

**Guidance**

When K is Active, the **next event** shall occur.

What if …each object had its own version history?

Product line has the option to introduce an “upgrade” vs a “variant”
Future State of the Practice

• “Information/ Model Centric Configuration Management”
  – Unintentional “versions” can be screened by sharing projects
    • Change does not affect current project definition/ baseline identification
    • Project leads screen for “upgrades” and apply changes as needed ...
    • ...or suggest a variant introduction if original version still applies
  – Each project always maintains “my project” view of correct objects
  – Features can be introduced by a product team without perturbing active projects.
Future State of the Practice

“Information/ Model Centric Configuration Management”

Common FMS System Reqts

KC-135 V1.0

Navigation

When $X$ is True, the active waypoint shall be displayed with accents.

When $X$ is False, the active waypoint shall be displayed with no accents.

When $Y$ is less than 0, the future waypoints shall be displayed in sequence.

Guidance

When $K$ is Active, the next event shall occur.
### Future State of the Practice

#### Information/Model Centric Configuration Management

<table>
<thead>
<tr>
<th>Common FMS System Reqts</th>
<th>C-130 V1.0</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Navigation</strong></td>
<td></td>
</tr>
<tr>
<td>R34</td>
<td>When X is True and Y is FALSE, the <em>active waypoint</em> shall be displayed with accents.</td>
</tr>
<tr>
<td>R35</td>
<td>When X is False, the <em>active waypoint</em> shall be displayed with no accents.</td>
</tr>
<tr>
<td>R36</td>
<td>When Y is less than 0, the <em>future waypoints</em> shall be displayed in sequence.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Guidance</strong></th>
<th>V2.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>When K is Active, the <em>next event</em> shall occur.</td>
<td></td>
</tr>
</tbody>
</table>

© Copyright 2010 Rockwell Collins, Inc.  
All rights reserved.
Current State of the Practice

• “Document-Centric Reallocation”

FMS System Reqs

<table>
<thead>
<tr>
<th>Navigation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product Requirement A</td>
</tr>
<tr>
<td>Product Requirement B</td>
</tr>
</tbody>
</table>

MMS System Reqs

<table>
<thead>
<tr>
<th>Navigation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product Requirement A</td>
</tr>
<tr>
<td>Product Requirement B</td>
</tr>
</tbody>
</table>

..but what if we need to reallocate functionality?

Today – same as “Clone and Own”!

Each “product” is expected to maintain it’s own “document”.

© Copyright 2010 Rockwell Collins, Inc. All rights reserved.
Future State of the Practice

- Consider the following:

<table>
<thead>
<tr>
<th>Paragraph</th>
<th>Activity/ Function f(x)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Navigation</td>
<td>Navigate</td>
</tr>
</tbody>
</table>

When X is True, the **active waypoint** shall be displayed with accents.

When Y is less than 0, the **future waypoints** shall be displayed in sequence.

When X is False, the **active waypoint** shall be displayed with no accents.
Future State of the Practice

- Reallocation

When $X$ is True, the **active waypoint** shall be displayed with accents.

When $Y$ is less than 0, the **future waypoints** shall be displayed in sequence.

When $X$ is False, the **active waypoint** shall be displayed with no accents.
Future State of the Practice

- Documentation is Preserved!

**FMS**

Navigate

- When $X$ is True, the **active waypoint** shall be displayed with accents.
- When $Y$ is less than 0, the **future waypoints** shall be displayed in sequence.
- When $X$ is False, the **active waypoint** shall be displayed with no accents.

**KC-135 FMS System Reqts**

Navigate

- When $X$ is True, the **active waypoint** shall be displayed with accents.
- When $Y$ is less than 0, the **future waypoints** shall be displayed in sequence.
- When $X$ is False, the **active waypoint** shall be displayed with no accents.

Guide

© Copyright 2010 Rockwell Collins, Inc. All rights reserved.
...connect each discrete piece of information together within the context of the a model ...
Future State of Information/Model Mgt

...combined with object level configuration management ...
So, Why Aren’t We Doing This NOW?

- What does it mean to “share”:
  - one element on a diagram?
  - one component from an architecture?
  - one requirement?
  - one interface?

- What if 99% of product was common? How do you manage the 1% variation - without Copy/Paste?

- Product Family Engineering technology must address “one” before it can address “many”
Existing Tools – Document Centric Structures!

- **DOORS**
  - Module ≈ Document
  - Module under Configuration Control
  - All objects baselined for each document version
  - No access to “historical” versions

- **Enterprise Architect**
  - Package ≈ Document
  - All items in a Package under version control
  - Cannot share “one” item between projects
  - Project sharing of same EAP **file** would be difficult
  - Note: Not even EA “drawings” have separate version control within EA.

- Current technology stores things in whole collections (files) and not as individual objects
Current State of Tools

- Short-comings of current tools and associated practices
  - Unable to ensure baselined projects are unintentionally modified by projects in progress
    - Lack robust enough configuration control capability
  - Unable to provide change identification between own project baselined documents
    - Since each project baselines DOORS modules as needed, other project baselines are reset, so publishing with meaningful change bars is unavailable at this time
  - Rearchitecting results in loss of commonality
    - Since DOORS is not fully connected to an architectural model, when a function is “reallocated” to a new component in a project architecture, those requirements are moved to a new “document” in DOORS, resulting in lost commonality
  - Lack of effective access controls
    - Foreign nationals are barred from the shared environment since all projects are always in view
    - Lacks ability to isolate users to only specific project views
  - Lack of object-level configuration management
    - “Objects” are stored in collections (files) – not as distinct objects
    - Current CM conventions still operate on macro “files”, and not on “micro” objects
Vision of the Future

- Information object repository - distributed
  - Distributed library
  - Config Mgt broker
  - Object Library Schema
  - Plug-In modules to exercise library objects
  - Custom HMI configuration
Raymond W. Jorgensen
Rockwell Collins
MS 108-265
400 Collins Rd NE
Cedar Rapids, IA 52498
rwjorgen@rockwellcollins.com
(319)295-2615