DOORS Requirement Reuse for Concurrent Development

Presented By:
Scott Edinger
Medtronic, Inc
scott.edinger@medtronic.com
scott.edinger@ieee.org
Agenda

- Framing the Problem: Concurrent Development
- Reuse Solution
  - Part 1: Release Content Sharing
  - Part 2: Project Variation Within a Release
  - Part 3: Component Content Sharing
- Putting it all together
- Demo
Brief Overview of Concurrent Development Challenges

- Changes in one project need to be propagated into all future projects
- Projects are in different stages of being developed by different teams
- Requirements architecture and structure may change between projects
- Effort is expended tracking changes and manually synchronizing from one project to another
Release Development

- **Release**
  - Name given to one or more projects that develop a major version of product software or firmware
  - Releases build upon requirement content from previous releases
  - At any one time, there might be several active releases, each in a different stage of development life-cycle (DLC)

  ![Diagram showing relationships between Tau, Sigma, and Upsilon releases]
Project Iterations Within Releases

- Releases are made up of one or more development project iterations
  - Functional iterations
    - Usually named by the functionality that is added
  - Point releases
- Project iterations converge and share a common architecture and structure within the same release
- Development life-cycle (DLC) processes:
  - Consistent within the release
  - Based on the state of the release
Moving Projects Between Releases

- Project iterations need to be able to move from one release to another:
  - Functionality may be delayed to a later release
  - May need to coordinate with some other part of the delivered system
  - May need to be pulled into a previous release to meet a critical customer need
- Project inheritance needs to be adjusted
- Traceability links need to be moved or adjusted
Functional Project Development

- Functional Projects
  - Name given to one or more projects that develop functionality that hasn’t yet been assigned to a release
  - Functional projects build from a subset of requirement content from last release
  - At any one time, there several pilot projects in play
  - DLC is much less constrained than a release
  - Functional project is merged back into a derived release
Statement of Reuse Solution

● Opportunity
  ● DOORS “out-of-the-box”
    ● Provides solid foundational infrastructure to build customized, sophisticated control mechanisms
    ● Offers limited mechanisms and tools for handling concurrent development

● Solution
  ● Medtronic has developed a framework and DOORS add-in mechanism that implements several approaches of managing requirement reuse for concurrent project development
Requirement Reuse Solution

- **Part 1. Reuse Between Releases**
  - Used to share content between releases developed concurrently

- **Part 2. Project Variation Within a Release**
  - Used to manage project iterations within a single release

- **Part 3. Component Reuse Within a Release**
  - Used to replicate one or more instances of common requirement content within a single release
Reuse Solution Part 1: Reuse Between Releases
Reuse Solution Part 1: Desired Outcomes

- To reuse a previous project requirement content in a follow-on, derived project
- A Derivative Project
  - Is derived from a previous project
  - Can be worked at the same time as the previous project
  - Can be changed, but will not affect the previous project
- Change Management
  - Can easily see changes from the previous releases
  - Has automated mechanism to merge changes (inserts, updates, deletes) from the previous release
- Freedom to Change with Design
  - Derived project objects can be reorganized and restructured from the previous project, even moved into different modules
Reuse Solution Part 1: Challenges

- What do I reuse between projects/releases?
  - What types of objects?
    - Sections?
    - Features?
  - What Attributes?
  - What Links?
  - What changes do I propagate?
    - Deletes?
    - Moves?

- How does development life-cycle (DLC) affect reuse?
  - Derived projects need fewer constraints than previous mature projects
  - Reuse mechanism needs to be configurable to DLC need

- DOORS
  - Has limited mechanisms for handling concurrent development
    - Copy+link
    - Suspect links (becomes overloaded by too many links)
    - Requires manual synchronization and link maintenance
  - Module object ID scheme does not handle concurrency
    - Once a module is copied, the object IDs within the two modules diverge
  - Copying module replicates links to previous project data (undesirable)
  - Object history is tied to a module—moving between modules loses history
Reuse Solution Part 1: Basic Idea

- Requirement module is copied into the derived release folder
- Identical objects are linked using “Derived Project” Link
- Shared content is “Locked” to prevent edit
- Derived release is regularly synchronized
- Uses custom global reference ID mechanism to identify matching content in derived releases
Global Reference ID

- Reference ID is dynamically generated from multiple components (similar to DOORS identifier)
- Format:
  - {Mod Prefix}{Info Type Prefix}{Ref #}-{Variation #}

<table>
<thead>
<tr>
<th>Mod Prefix</th>
<th>Module attribute: Reference ID Prefix</th>
<th>Specifies the module prefix to put in the reference ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Info Type Prefix</td>
<td>Object Attribute: Information Type</td>
<td>Used to lookup a short name using a global DxlObject hash of constants</td>
</tr>
<tr>
<td>Ref #</td>
<td>Object Attribute: Reference Number</td>
<td>ID used to identify an object across releases</td>
</tr>
<tr>
<td>Variation #</td>
<td>Object Attribute: Reference Variation</td>
<td>Variation number used to uniquely identify a version of a requirement within a module</td>
</tr>
</tbody>
</table>
Reuse Solution Part 1: Object Derived States

- **Shared**
  - Edit not allowed--Locked
  - Shared content will be synchronized from previous release

- **Branched**
  - Edit is allowed
  - Shared content is not synchronized from previous release

- **Removed**
  - Object was removed from previous release
  - Synchronization will clear “Project Label” attribute
  - Not physically deleted—can still be managed

- **Orphaned**
  - Object was branched and then removed from previous release
  - Synchronization will not remove from current release
# Reuse Solution Part 1: Content Sharing Attributes

## Module Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shared Content Attributes</td>
<td>Specifies the object attributes that are “Shared” and are acted upon by edit protection triggers and content synchronization</td>
</tr>
<tr>
<td>Shared Content Options</td>
<td>Used to configure the what features of the shared content mechanism are enabled</td>
</tr>
</tbody>
</table>

## Object Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
</table>
| Derived From               | XML text containing:  
  - What kind of derived relationship exists  
  - State of the derived relationship  
  - “Project Label” an object is derived from |
| Synchronize Action         | Synchronization actions that have been performed on an object               |
| Synchronize Date           | Last date a synchronization action was performed on an object               |
| Synchronize Log            | Detailed log of all synchronization actions that have been performed on an object |
Reuse Solution Part 1: Content Sharing Synchronization

- Synchronization process
  - Inserts new content, relative to other objects around it
  - Updates shared content
  - Sets object as “Removed” when object was removed from the previous release
  - Does not synchronize “Sections”—allowing structure to change between releases

- Runs in nightly batch or interactively from module menu
Reuse Solution Part 1:
Reuse Menu Operations

- Branch Content From Previous Release
  - Changes the derived state from “Shared” to “Branched” (or “Removed” to “Orphaned”)
  - Allows shared attributes to be edited
- Restore Content Sharing With Previous Release
  - Changes the derived state from “Branched” to “Shared” (or “Orphaned” to “Removed”)
  - Replaces the content of the shared attributes with the content of the previous release
- Synchronize Shared Content in Object/Section/Module
  - Synchronizes the shared attributes of an object, section or module with content from a previous release
- Delete Unused Objects in Section/Module
  - Safely deletes objects that are no longer associated with a project
  - Only deletes when not referenced by links
  - Removes the “Derived Project” link (if not referenced by a follow-on, derived release)
Reuse Solution Part 1: Visibility of Reuse Changes

- Each module has a view that shows “Changes From the Previous Release”
  - Shows Inserts, Updates, Removed, and Moves
  - Updates to shared object attributes are shown with markups
- Easy and quick to see release differences in real-time
Reuse Solution Part 2: Project Variation Within a Release
Reuse Solution Part 2: Desired Outcomes

- Need to be able to create variations of requirement content within the same module
- Variations
  - Can easily be identified for the same requirement reference
  - Can be shared by more than one project
  - Can be locked to force new changes into a new variation
- Branching
  - A “Branching” mechanism creates a related replica of a selected object variation
- View by Project
  - Allows viewing or publishing of all data elements associated with a specific project
Reuse Solution Part 2: Challenges

- Branching hierarchies of information is difficult
  - Branching child objects? Sections? Tables?
  - Do links get branched?
  - What are the rules?

- Process Issues
  - Project label maintenance must be done consistently—but in practice has been difficult to perform consistently using manual editing
  - How do you “lock down” a project variation and force branching to occur?
  - Projects might interfere with another

- DOORS
  - Object ID scheme cannot be used for identifying variations of the same object
  - Using partitioning for shareable editing, there is contention between projects in the same release module
Reuse Solution Part 2: Basic Idea

- An object is “branched” by creating a duplicate sibling object
  - Child objects are automatically replicated
- Uses custom global ID mechanism to identify an object and its variations
  - Contains variation number at end of ID
- Uses multi-valued “Project Label” attribute to identify the project variation
  - “Project Label” can only be manipulated via menu options
- Only “leaf” objects are allowed to be branched
  - No Sections, Features, etc
Reuse Solution Part 2: Reuse Menu Options

- **Branch Object**
  - Creates a new variation of the selected object
  - Project labels for the new branch are moved

- **Add Project Label to Object**
  - Associates an object variation with a project

- **Remove Project Label from Object**
  - Disassociates an object variation from a project

- **Move Project Label to Object**
  - Moves project labels from other variations to the selected object variation
Reuse Solution Part 3: Component Reuse Within a Release
Reuse Solution Part 3: Desired Outcomes

- Replicate a requirement or collection of requirements in more than one module location

- A Common Component
  - Is treated as a single entity, but can be made up of several objects
  - Can only be edited in one source location
Reuse Solution Part 3: Basic Idea

- Common component object is inserted into target location
- Object is linked from a common component module using a “Common Requirement” link
- Shared content is synchronized from common component module to target module
- Child objects are replicated
Reuse Solution Part 3: Reuse Menu Options

- Synchronize Shared Content in Object/Section/Module
  - Synchronizes the shared attributes of an object, section or module with content from the linked common component
Putting It All Together: Parts 1 and 2

Use of Project Labels

- Project labels on each object
  - Previous release label
  - One or more project iteration labels in between
  - Current release label
- Previous release and current release labels are “bookends”
- Usually latest project label and current release label are kept together
Putting It All Together: Parts 1 and 3

- Common requirements are derived from previous release common requirements
- Synchronization acts on the full integration of reuse parts 1 and 3
- Common requirement modules are synchronized before modules that use them
Benefits of this Reuse Solution

- Removes manual re-entry and synchronization of requirements between project and release iterations
  - Quality is increased
  - Time is saved
  - Communication between development efforts is improved
- Bottom Line:
  - Critical to effectively develop products concurrently
  - Significant value is added to DOORS
Highlights of this Reuse Solution

- Uses Standard DOORS Functionality
  - Uses DOORS Links
    - Can use traceability functionality
  - Uses standard DOORS add-in mechanisms
    - Can easily port from one version of DOORS to the next (using the full DOORS client application)
- Multi-user friendly
  - Operates in Shareable Edit mode
Negative Side-Effects - 1

- Views developed with the Trace Wizard are not portable between derived projects

Solutions:
1. Make module branching (copy & link) code adjust views dynamically
2. Replace trace wizard views with code that is reuse friendly
Suspect Links

- Built-in Suspect Links Mechanism is rendered useless
- Introduction of sizeable numbers of reuse links triggers suspect links from CreateLink and DeleteLink history events

Solution:
- Replace the suspect link mechanism with something more flexible
- Demo
Reuse Strategy for Test Content

- Validation and verification testing modules
  - Only use part 1 of the reuse solution
  - Project label maintenance has been removed
    - Does not add enough value for the effort
  - Traceability linking is more important for defining project test coverage and completeness
- Easier to move to other non-DOORS test management solutions (RQM, etc)
  - They only use traceability links to define project test coverage
Moving projects between releases
- Simple project merging has been done, but full merge mechanism not yet developed
- Change history will be “locked” in the release module you are moving “from”
  - Can only copy the project object attributes but history cannot be moved with the object
- Architecture of DOORS will constrain design
Remaining Challenges - 2

- Synchronization
  - Moves
  - Trace links
  - Common Requirement Components

- Change Reporting
  - Showing changes on OLE objects (ie. Word table markups)
  - Showing changes on Common Requirement Components
Questions?