Spacecraft Flight Software Design Patterns Discovery

Michael Phillips – Lockheed Martin Fellow, R&D Principal Investigator
Amy Mok – Lockheed Martin, R&D Technical Lead
Agenda

• Overview
  – Business Case
  – Introduction of the R&D Project
• COT Tools for Pattern Detection
  – Diagrams
  – Metrics
  – Sample Usage
• Sample Results
• Obstacles & Challenges
• Summary
2. Software Estimates for code reuse (Modified + Untouched) migrates rapidly from ReUse (100% untouched) to Modified to New during a program lifecycle.
Discover Common Design Patterns across existing TRL9 Spacecraft Flight Software using COTS Tools. Implement, Test, and Deliver TRL 4 FSW Modules for FSW Reference Designs in a few domain areas commonly re-designed and re-implemented during programs.

What is a Design Pattern?

• A software pattern is a portion of a end product that is repeated or replicated in multiple SW products, where the software products may be modeling products and or source code products.

• Patterns are architected into software products and or introduced independently during software design when software engineers choose similar or identical solutions to implement similar functionality.

Task 1: FSW Design Commonality Discovery

Task 2: Develop Best Practice SW Reference Designs

Task 3: Implement, Test, and Deliver TRL 4 FSW Modules

Reference Designs for future SW product lines using COTS tools for design pattern discovery
Types of Software Patterns

1. Architecture Patterns:
   - Patterns applied across all software domains
     • “Base” classes used or extended by each domain
     • Patterns for creating constants, parameters and variables
     • Patterns for creating command and telemetry messages
     • Patterns for creating initialization and processing threads
     • Patterns for fault management

2. Functional Patterns:
   - Patterns identified within multiple software baselines
     • Requirements common to multiple software designs
     • Functionality common to multiple software designs

3. Logic and Algorithm Patterns:
   - Patterns identified within multiple software baselines
     • Processing threads common to multiple software designs
     • Algorithms common to multiple software designs
     • Equations common to multiple software designs

4. Implementation Patterns:
   - Patterns applied within one or more software domain
     • Modeling or coding solutions to implement logic, algorithms, and equations
     • C++ Templates / Ada Generics
     • Generalization & Extension
Use Cases & 2013 Pilots

Reverse Engineering to Support Program Needs (Immediate)

SWE Central Function for Enterprise Commonality

The R&D return is achieved through shorter software development cycle and increase of design reuse (i.e., commonality)
# Tools Selected to aid in Pattern Detection

<table>
<thead>
<tr>
<th>Tool Name</th>
<th>Vendor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Axivion Bauhaus Suite</td>
<td>Axivion</td>
</tr>
<tr>
<td>Together</td>
<td>Borland</td>
</tr>
<tr>
<td>Imagix 4D</td>
<td>Imagix</td>
</tr>
<tr>
<td>LDRA Testbed</td>
<td>LDRA Software Technology</td>
</tr>
<tr>
<td>McCabe IQ</td>
<td>McCabe Software</td>
</tr>
<tr>
<td>C/C++ Test</td>
<td>Parasoft</td>
</tr>
<tr>
<td>Ptidej</td>
<td>FOSS</td>
</tr>
<tr>
<td>Sotoarc / Sotograph</td>
<td>Hello2morrow</td>
</tr>
<tr>
<td>UMLStudio</td>
<td>Pragsoft</td>
</tr>
<tr>
<td>Understand</td>
<td>SciTools</td>
</tr>
<tr>
<td>Visual Paradigm</td>
<td>Visual Paradigm</td>
</tr>
<tr>
<td>Rhapsody</td>
<td>IBM</td>
</tr>
</tbody>
</table>
## Tool Capabilities Matrix

<table>
<thead>
<tr>
<th>Tool</th>
<th>Architecture</th>
<th>Data Structures</th>
<th>Reverse Engineering</th>
<th>Metrics</th>
<th>Supported Languages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Together</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>ADA</td>
</tr>
<tr>
<td>Imagix4D</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>C</td>
</tr>
<tr>
<td>LDRA</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>C</td>
</tr>
<tr>
<td>McCabe IQ</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>C</td>
</tr>
<tr>
<td>C/C++ Test</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>C</td>
</tr>
<tr>
<td>UMLStudio</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>ADA</td>
</tr>
<tr>
<td>Ptidej</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>ADA</td>
</tr>
<tr>
<td>Sotoarc / Sotograph</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td>ADA</td>
</tr>
<tr>
<td>Understand</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>ADA</td>
</tr>
<tr>
<td>Visual Paradigm</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>ADA</td>
</tr>
<tr>
<td>Rhapsody</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>ADA</td>
</tr>
</tbody>
</table>
Metrics Show:

- Indication of which code baselines should be used to develop the reference design
- Quantitative differences between the multiple code baselines
- Assurance of the quality of the reference design that will be distributed

Some Examples:

- Complexity (Cyclomatic, Essential, Halstead)
  - Difficulty in understanding, implementing, and testing decision logic
- (Lack of) Cohesion
  - How related the functions and functionality of a module are.
- Coupling
  - Degree to which each module relies on other modules
- Depth of Inheritance
  - How deeply modules inherit from each other
- SLOC
  - Amount of code necessary to implement functionality
- Many more...
End-To-End Process

Inputs:
- Source Code
- Models
- Requirements
- CDRL Documentation
- Other Documentation

Use Metrics:
Characterize individual patterns
Use Trade Study Tool:
To compare different patterns for the same functions

Select Tools

Discover Patterns

Evaluate Patterns

Implement Patterns

Deploy Patterns

Source Code Analysis Tool Suite

All Patterns Identified

Preferred Coded Patterns

Preferred Modeled Patterns

CM’ed Pattern Product Pkgs.
Sample Detailed Design – Class Diagrams

Detailed Design

Reverse Engineered Design

End User Artifacts: SW Architecture Diagrams - Differences between planned and actual can be discovered providing users with correct design for review and implementation
End User Artifacts: SW Logic/Algorithm Diagrams - Represents actual, detailed, unambiguous, and straightforward views of the software design
Sample Requirement Traceability

- SRS0001 – FSW shall ...
- SRS0002 – FSW shall ...
- SRS0003 – FSW shall ...

End User Artifacts: Mapping to SW Requirements – Ability to easily map requirements to design, allowing program to determine new, modified, or reuse of SW design
Program Usage of Functionalities

Functionalities that most programs had implemented represent critical functions.
Functionalities that only a few programs had implemented represent mission-unique functions.
Obstacles & Challenges

• Usually difficult to obtain FSW and FSW Design Artifacts from heritage programs
  – Each program has their own process, priorities, time table, and restrictions

• Creating common designs can be tedious (the tools help greatly, but it’s still a manual process)

• Program-specific design decisions must be filtered out from the common design

• Lots of interest for derived designs of a single heritage program, but need more interest in the designs derived from multiple heritage programs

• Comparing different designs can be subjective and difficult as each design has different capabilities
Summary

- We believe there is significant room to improve the productivity of the Flight Software Team by leveraging design reuse from existing programs.

- COTS tools have a tremendous capability to efficiently reverse-engineer both software design and metrics that can augment program design artifacts.
  - These tools have limitations that require both processes and manual effort to effectively use the artifacts produced by the tools.
  - In some cases, the reverse-engineered design is more factual than the existing design artifacts, since it was derived for the actual implementation, not the planned implementation.

- Providing reference design information to both individual contributors for specific SW domains, as well FSW technical leads and managers, needs to be organizationally institutionalized in order to effectively improve productivity on programs.