Verification and Validation of Models for Embedded Software Development

Prashant Hegde
MathWorks India Pvt. Ltd.
Designing complex systems

I know what I want to build

I built what I wanted

Have I implemented what I wanted? Or Missing something?

Are there any hidden design errors?

Is there something I don’t know about what I want?
Verification and Validation Challenges

- Traceability: Requirement <-> Model <-> Code
- Test vector creation: functional and structural
- Coverage: Modified Condition and Decision Coverage, Condition, Decision
- Test Reports
Agenda

- Perform early requirements validation
- Review models against modelling standards and guidelines
- Prove the integrity of functional requirements
- Perform coverage analysis
- Detect design errors in a model
- Generate test cases for design and functional requirements
- Verification and Validation workflow for Industry Standards
Why to Verify and Validate the models?

- Structural and Algorithmic complexities in models
- Model-Based Design simplifies reuse
Key Model Verification and Validation Techniques

- Track Design Changes: Traceability Analysis
- Conformance: Modeling Standards Checks
- Functional Testing: Functional Behaviour
- Structural Testing: Model Coverage
- Formal Verification:
  - Design Error Detection
  - Test Generation
Agenda

- Perform early requirements validation
  - Review models against modelling standards and guidelines
  - Prove the integrity of functional requirements
  - Perform coverage analysis
  - Detect design errors in a model
  - Generate test cases for design and functional requirements
  - Verification and Validation workflow for Industry Standards
Requirement Management Interface+ Report

Requirements linking and traceability

- Bi-directional linking with external documents
  - For Simulink and Stateflow
  - Requirements consistency checks
  - Extensibility API
  - Report generation

- DOORS integration
  - Linking with read-only requirement documents

- Embedded Coder integration
  - Embeds requirements as comments in source code
Traceability: Requirements ↔ Model

Requirements Management Interface

- Creating links between textual documents and model objects

1.3 Resume Speed/Accelerate Button
The controller shall have an input button to:
- set the target speed to last acceptable target speed when the cruise engaged (active)
- accelerate (increase) the target speed when the cruise control is engaged

2 Functional Requirements
2.1 Disabled (off) during start-up
Initial state of cruise control system shall be disabled.

2.2 Not engaged with enabling (on)
The cruise control system shall not be initially engaged with enabling.

2.3 Disengaged (not active) when disabled (off)
The cruise control system shall disengage with disabling.

2.4 Initial transition from disengaged (inactive) to engaged (active)
"Set Speed" input after enabling (on)
The cruise control shall only transition to engaged (active) the first time after has been enabled with a "Set Speed" input. Target speed will be set to e
Requirements Traceability Report for CruiseControl

Table of Contents

1. Model Information for "CruiseControl"
2. Traceability Summary for "CruiseControl"
3. Chart - Compute target speed

List of Tables

1.1. CruiseControl
2.1. Artifacts linked in model
3.1. Objects in "Compute target speed" that have requirements

Chapter 1. Model Information for "CruiseControl"

Table 1.1. CruiseControl

<table>
<thead>
<tr>
<th>ModelVersion</th>
<th>1.307</th>
<th>ConfigurationManager</th>
<th>None</th>
</tr>
</thead>
<tbody>
<tr>
<td>LastModifiedDate</td>
<td>Wed Feb 11 22:16:45 2015</td>
<td>LastModifiedBy</td>
<td>mreddy</td>
</tr>
</tbody>
</table>
Requirement Management Interface : Demo

Requirement Management Interface : Demo
Agenda

- Perform early requirements validation
- **Review models against modelling standards and guidelines**
  - Prove the integrity of functional requirements
  - Perform coverage analysis
  - Detect design errors in a model
  - Generate test cases for design and functional requirements
  - Verification and Validation workflow for Industry Standards
Model Advisor
Conformance to standards

- Model Advisor is used to
  - Enforce model standards and best practices
  - Detect and troubleshoot modeling and code generation issues
  - Check models for (a subset of) known version upgrade issues
Modelling Standards Checking

Verify model complies with modeling guidelines.

Tips
To enable or disable a check, select or clear the check box next to the check name.
To enable or disable all checks within a folder, right-click the folder and then click "Select All" or "Deselect All".
To run checks, select a folder or check in the left pane.
For a list of all possible actions, right-click an object in the left pane.
To show or hide By Product folder, select or clear "Show By Product Folder" in the Settings > Preferences dialog box.
To show or hide By Task folder, select or clear "Show By Task Folder" in the Settings > Preferences dialog box.

Legend
- Not Run
- Passed
- Failed
- Warning
- Running this check triggers an Update Diagram.
Model Advisor : Demo
Agenda

- Perform early requirements validation
- Review models against modelling standards and guidelines
- **Prove the integrity of functional requirements**
- Perform coverage analysis
- Detect design errors in a model
- Generate test cases for design and functional requirements
- Verification and Validation workflow for Industry Standards
Simulink Test: Develop, Manage and execute simulation based tests

<table>
<thead>
<tr>
<th>1. Test Harnesses</th>
<th>2. Test Sequence Block</th>
<th>3. Test Manager</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Synchronized, simulatable test environment</td>
<td>• Inputs and assessments based on logical, temporal conditions</td>
<td>• Author, execute, manage test cases • Review, export, report</td>
</tr>
</tbody>
</table>

![Test Harness](image1)

Main Model

Component under test

![Test Sequence](image2)

![Test Manager](image3)
Test Case Templates

1. Simulation Test
   - Input
   - Output
   - ✓ Assessment Criteria

2. Baseline Test
   - Input
   - Output
   - ✓ Assessment Criteria
   - File
   - Expected Outputs

3. Equivalence Test
   - Input
   - Output
   - ✓ Assessment Criteria
   - Input
   - Output
Simulink Test : Demo
Agenda

- Perform early requirements validation
- Review models against modelling standards and guidelines
- Prove the integrity of functional requirements

**Perform coverage analysis**
- Detect design errors in a model
- Generate test cases for design and functional requirements
- Verification and Validation workflow for Industry Standards
Model Coverage

- Execution analysis
  - Based on the model structure
  - Dynamic – data collected during simulation

- Coverage results
  - Displayed directly in the model
  - Available in a separate HTML report linked with the model objects

- Supports
  - Simulink
  - Stateflow
  - Embedded MATLAB
  - S-functions
Simulation Testing Workflow

- **Design**
- **Detect unreachable design logic**
- **Simulate and validate controller response**
- **Review functional behavior**
- **Structural coverage report**
Functional (Black Box) Testing

Test Cases

Signal Builder
Module Test-Harness

Test Cases (Signal Builder)

Model (Model Block)

Output Check (Assertion Block)
Model Coverage Analysis – Why is it needed

- Coverage metrics identifies untested portions of the model
Model coverage : Demo
Simulation Based Testing: Concerns

- Writing exhaustive requirement-based test cases
- Creating test simulation environment and generating reports
- Missing coverage in the model
- Unexpected behavior in model functionality
- How to address this problem?

Why do we have missing coverage?

Simulink Design Verifier
Generate Test Vector for Missing Coverage

Requirements → Component Models → Simulation → Missing Coverage → Why do we have missing coverage?

Generate Test Vector for Missing Coverage
Agenda

- Perform early requirements validation
- Review models against modelling standards and guidelines
- Prove the integrity of functional requirements
- Perform coverage analysis

- Detect design errors in a model
- Generate test cases for design and functional requirements
- Verification and Validation workflow for Industry Standards
Simulink Design Verifier

Property proving
- Generates a proof for a requirement
  - For example: Thrust reversers shall not deploy in flight
- Proves user defined model properties and generates reports
- Generate example of violations if applicable

Design error detection
- Detect two types of errors:
  - Integer data overflow
  - Division by zero
- Detect presence of errors and generates reports
- Generate example cases to demonstrate the errors

Test generation
- Find test vectors for coverage not achieved by functional tests
  - Find unspecified requirement
  - Identify derived requirement
  - Remove model function not traceable to a requirement, unless identified as derived
  - Detect unachievable conditions and decisions
  - Identify decisions that are not independent (do not meet MCDC coverage)
Identifying Design Errors Early

Automatic identification of hard-to-find design inconsistencies in the model

- Dead logic
- Division by zero
- Out of bound Array Access
Demo Simulink Design Verifier : Demo
Polyspace Products for Code Verification

- **Quality improvement**
  - Prove the absence of run-time errors in source code
  - Measure, improve, and control

- **Usage**
  - Simple colored source code
  - No compilation, no execution, no test cases
  - For C/C++ or Ada

- **Process**
  - Run early in development cycle
  - Use for automatically generated and handwritten code
PolySpace Workshop

- Meeting Room -2
PolySpace Workshop

- Hall - 6
Agenda

- Perform early requirements validation
- Review models against modelling standards and guidelines
- Prove the integrity of functional requirements
- Perform coverage analysis
- Detect design errors in a model
- Generate test cases for design and functional requirements

- Verification and Validation workflow for Industry Standards
# MathWorks Verification and Validation Product Portfolio

<table>
<thead>
<tr>
<th>Product</th>
<th>Capabilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simulink Test</td>
<td>Author, execute, and manage simulation-based tests for models and generated code</td>
</tr>
<tr>
<td>Simulink Verification &amp; Validation</td>
<td>Trace to requirements, check model standards, perform coverage analysis</td>
</tr>
<tr>
<td>Simulink Design Verifier</td>
<td>Identify design errors, automatically generate test vectors, verify designs against requirements</td>
</tr>
<tr>
<td>Report Generator</td>
<td>Design and generate reports from MATLAB applications</td>
</tr>
<tr>
<td>Polyspace Bug Finder</td>
<td>Find software bugs and check compliance to MISRA</td>
</tr>
<tr>
<td>Polyspace Code Prover</td>
<td>Prove the absence of run-time errors in software</td>
</tr>
<tr>
<td>Simulink Real-Time</td>
<td>Build, run, and test real-time applications</td>
</tr>
</tbody>
</table>
Thank you!