MATLAB EXPO 2019

Simplifying Requirements Based Verification with Model-Based Design

Vamshi Kumbham
Pilot Engineering
Key takeaways

- Verify and validate requirements earlier
- Identify inconsistencies in requirements by using unambiguous assessments
- Traceability from requirements to design and test

“By enabling us to analyze requirements quickly, reuse designs from previous products, and eliminate manual coding errors, Model-Based Design has reduced development times and enabled us to shorten schedules to meet the needs of our customers.”

- MyoungSuk Ko, LS Automotive
Challenge: Errors introduced early but found late

- Most errors introduced
- Unit test finds some errors
- Errors found during integration or in field

Requirements → Specification → C/C++ → Generated code
Cost of finding errors increases over time

Testing Cost

Time

Requirements → Specification → C/C++ → Generated code
Challenges with requirements based verification

Are all requirements implemented?

Is requirement interpreted correctly?

Is the implementation functioning correctly?

How to avoid modifying the design for test?
Simulink models for specification

Model-Based Design enables:

- Early testing to increase confidence in your design
- Delivery of higher quality software throughout the workflow
Multiple languages to describe complex systems
Ad-Hoc Testing: Explore behavior and design alternatives
Validate behavior earlier with simulation
Validate Behavior Earlier with Simulation
Complete Model Based Design

Simulink Models

- Requirements
- Design Model
- Model used for production code generation

Generated code

Code Generation

Simulink Models

- C/C++
- Generated code
Systematically verify requirements

Are all requirements implemented?
Is the implementation functioning correctly?
Are designs and requirements consistent?

Requirements Based Testing

Requirements
Design Model
Simulink Models
Model used for production code generation
C/C++
Generated code
Integrate with requirements tools and author requirements

- **Import from:**
  - Word / Excel
  - IBM® Rational® DOORS®
  - ReqIF™ standard

- **Update synchronizes changes from source**

- **Edit and add further details to import**

- **Author requirements**

- **Export ReqIF**
  - Enables roundtrip with external tools
Roundtrip workflow with external tools thru ReqIF

- Import from:
  - Word / Excel
  - IBM® Rational® DOORS®
  - ReqIF™ standard
- Update synchronizes changes from source
- Edit and add further details to import
- Author requirements
- Export ReqIF
  - Enables roundtrip with external tools

Requirements Managements Tools

External Requirements

• Import from:
  • Word / Excel
  • IBM® Rational® DOORS®
  • ReqIF™ standard
• Update synchronizes changes from source
• Edit and add further details to import
• Author requirements
• Export ReqIF
  • Enables roundtrip with external tools
Requirements Verification with Simulink

Requirements

- TransmissionReq
  - 1
    - 1.1: Transmission Operating Modes
      - Reverse cannot be entered from drive
    - 1.2: Engine only starts in Park

Implemented By

Simulink / Stateflow

Verified By

MATLAB Unit Test

Test Case

Inputs

- MAT / Excel file (input)
- Signal Editor
- Test Sequence

Assessments

- MAT / Excel File (baseline)
- Test Assessments

Test Harness

Simulink Test
Requirements Verification with Simulink

**Test Case**

**Inputs**
- MAT / Excel file (input)
- Signal Editor
- Test Sequence

**Requirements**

- Driver Switch Request Handling
  - 1.1 Switch precedence
  - 1.2 Avoid repeating commands

**Implemented By**

**Verified By**

**Simulink / Stateflow**

**Implemented**

**Verified**

- MATLAB Unit Test
- Simulink / Stateflow
- MATLAB / Excel file (baseline)
- Test Assessments

**Assessments**

**MATLAB EXPO**
1 Requirements for the basic Heatpump Controller

1.1 Idle when Temperature in Range
If the temperature difference is less than 1 degrees, the system shall be idle with all signals off.

1.2 Activate Fan
The fan shall activate when the temperature difference is greater than or equal to 1 degrees.

1.3 Activate Heat Pump
The pump shall activate when the temperature difference is greater than or equal to 2 degrees for more than 2 seconds and stay active for at least 2 seconds.

1.3.1 Cool Mode
If the room temperature is greater than the set temperature, the system shall cool the space.

1.3.2 Heat Mode
If the room temperature is less than the set temperature, the system shall heat the space.

1.4 Max Temperature
The difference between the room temperature and the set temperature should never exceed 6 degrees.

**Requirements in DOORS**
Example: Heat Pump Controller Implementation
Link requirements to implementation in model
Work with Model and Requirements with Requirements Perspective

Browser

Requirement Annotations

Badges

Implementation and Verification Status

Property Inspector
Isolate Component Under Test with Test Harness

House Heating System
1. Plot temperature of wall, window, and roof (see code)
2. Plot heat flow through wall, window, and roof (see code)
3. Explore simulation results using ssexplore
4. Learn more about this example
### Test Sequence Block: Step-based and temporal test sequences

<table>
<thead>
<tr>
<th>Symbols</th>
<th>Step</th>
<th>Transition</th>
<th>Next Step</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input</strong></td>
<td>Initialize</td>
<td>1. true</td>
<td>Cold_Outside</td>
</tr>
<tr>
<td>1. control_out</td>
<td>%% Initialize data inputs.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Tset = 23;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Troom_in = 23;</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Output</strong></td>
<td>Cold_Outside</td>
<td>1. Troom_in &lt;= 15</td>
<td>Hot_Outside</td>
</tr>
<tr>
<td>1. Tset</td>
<td>%% Check heating mode</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Troom_in = 23 - ramp(0,2);</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2. Troom_in</td>
<td>Hot_Outside</td>
<td>1. Troom_in &gt;= 27</td>
<td>Return_Idle</td>
</tr>
<tr>
<td>{</td>
<td>%% Check cooling mode</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Troom_in = 23 + ramp(0,2);</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Local</strong></td>
<td>Return_Idle</td>
<td>1. Troom_in &lt;= 22</td>
<td>End</td>
</tr>
<tr>
<td><strong>Constant</strong></td>
<td>%% Return to idle mode</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Troom_in = Troom_in-ramp(0,2);</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Parameter</strong></td>
<td>End</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Data Store Memory</strong></td>
<td>Troom_in = 22</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Activate Heat Pump

If the temperature difference exceeds 2 degrees for more than 2 seconds, then the pump shall activate for at least 2 seconds.

\[(|x_1 - x_2| \geq x_3)^\epsilon \land \square_{[0,t_1]}(|x_1 - x_2| \geq x_3) \rightarrow \square_{[0,t_2]} x_4\]
Author temporal assessments using form based editor
Execute assessments to verify requirements
Locate implementation of requirement using link
Translate textual requirements into unambiguous Temporal Assessments

- Compose assessments using form based editor
- View assessments as English-like sentence
- Review and debug temporal assessment results
- Link to requirements
Track Implementation and Verification

Implementation Status
- Implemented
- Justified
- Missing

Verification Status
- Passed
- Failed
- Unexecuted
- Missing
Observers: Separate test/verification logic from design

- Access nested signals without signal lines or changing dynamic response
- Avoid modifying interface for testing
- Simplify design and test by avoiding additional signal lines
Observers: Separate test/verification logic from design

- Access nested signals without signal lines or changing dynamic response
- Avoid modifying interface for testing
- Simplify design and test by avoiding additional signal lines
LS Automotive Reduces Development Time for Automotive Component Software with Model-Based Design

Challenge
Shorten development times for embedded control software used in automotive switches and components

Solution
Use Model-Based Design to model controller designs, run simulations, verify customer specifications, and generate error-free production code

Results
- Specification errors detected early
- Proven development approach established
- 80% Coding errors eliminated

“By enabling us to analyze requirements quickly, reuse designs from previous products, and eliminate manual coding errors, Model-Based Design has reduced development times and enabled us to shorten schedules to meet the needs of our customers.”
- MyoungSuk Ko, LS Automotive

Link to user story
MATLAB EXPO 2019
Summary

- Verify and validate requirements earlier
- Identify inconsistencies in requirements by using unambiguous assessments
- Traceability from requirements to design and test
Learn More

Key products covered in this presentation:
- Simulink Requirements
- Simulink Test

Learn more at Verification, Validation and Test Solution Page:
mathworks.com/solutions/verification-validation.html