Simplifying Requirements Based Verification with Model-Based Design

Dr. Tjorben Groß, Senior Application Engineer
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
Cost of finding errors increases over time

Testing Cost vs. Time

- Requirements
- Specification
- C/C++
- Hand 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

Code Generation

C/C++

Generated code
Systematically verify requirements

Are all requirements implemented?

Is the implementation functioning correctly?

Are designs and requirements consistent?

Requirements Based Testing

Simulink Models

Requirements → Design Model → 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 Verification with Simulink

Requirements

Transmission Operating Modes
1. Reverse cannot be entered from drive
2. Engine only starts in Park

Test Case

Implemented By

Test Harness

Verified By

Simulink / Stateflow

Inputs

MAT / Excel file (input)
Signal Editor
Test Sequence

Assessments

MAT / Excel File (baseline)
Test Assessments

MATLAB Unit Test

Test Harness

Simulink Test
Requirements Verification with Simulink

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

Implemented By

Verified By

Test Case
- Inputs
  - Scenario Signal 1
  - MAT / Excel file (input)
  - Signal Editor
  - Test Sequence

Simulink / Stateflow

Implemented

Verified

MATLAB Test Assessments
- MATLAB Unit Test
- MATLAB / Excel File (baseline)
- Simulink Test
- MAT / Excel file (input)

Assessments
- Test
- MATLAB / Excel file (input)

Implemented: 16, Justified: 0, None: 2, Total: 18
Example: Verifying Heat Pump Controller Requirements

1 Requirements for the basic Heatpump Controller

Temperature difference is defined as the difference between the room and the set temperature. The controller shall turn the fan on when the temperature difference has reached a certain level, to circulate the air. The controller shall turn the heatpump on when the temperature difference has reached another level, to heat or cool the space.

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
Example: Heat Pump Controller Implementation
Link requirements to implementation in model

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:

\[ \text{sign} \left( T_{\text{req}} - T_{\text{meas}} \right) = -1 \]

Implement:

- **COOLING**
  - entry:
    - fan_cmd = 1;
    - pump_cmd = 1;
    - pump_dir = 1;

- **HEATING**
  - entry:
    - fan_cmd = 1;
    - pump_cmd = 1;
    - pump_dir = -1;

After 2 seconds:

\[ \text{mag} \left( T_{\text{req}} - T_{\text{meas}} \right) \geq 2 \]
Work with Model and Requirements with Requirements Perspective

Requirement Annotations

Badges

Implementation and Verification Status

Browser

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 sscexplore
4. Learn more about this example
Test Sequence Block: Step-based and temporal test sequences
Test Assessments: Formalize and execute requirements

**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.

**Simple concept**

When <condition 1> is true,  
Then <condition 2> must be true for some time

**Hard to formalize**

\[(|x_1 - x_2| \geq x_3)^\varepsilon \land \Box_{[0,t_1]}(|x_1 - x_2| \geq x_3) \rightarrow \Box_{[0,t_2]} x_4\]

MTL logic
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

<table>
<thead>
<tr>
<th>Index</th>
<th>ID</th>
<th>Summary</th>
<th>Implemented</th>
<th>Verified</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>#1</td>
<td>Driver Switch Request Handling</td>
<td>Implemented</td>
<td>Passed</td>
</tr>
<tr>
<td>1.1</td>
<td>#2</td>
<td>Switch precedence</td>
<td>Justified</td>
<td>Failed</td>
</tr>
<tr>
<td>1.2</td>
<td>#3</td>
<td>Avoid repeating commands</td>
<td>Implemented</td>
<td>Passed</td>
</tr>
<tr>
<td>1.3</td>
<td>#4</td>
<td>Long Switch recognition</td>
<td>Implemented</td>
<td>Passed</td>
</tr>
<tr>
<td>1.4</td>
<td>#7</td>
<td>Cancel Switch Detection</td>
<td>Implemented</td>
<td>Passed</td>
</tr>
<tr>
<td>1.5</td>
<td>#8</td>
<td>Set Switch Detection</td>
<td>Implemented</td>
<td>Passed</td>
</tr>
<tr>
<td>1.6</td>
<td>#9</td>
<td>Enable Switch Detection</td>
<td>Implemented</td>
<td>Passed</td>
</tr>
</tbody>
</table>

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
Re-use tests developed for model to test code

Software in the Loop (SIL)
- Show functional equivalence, model to code
- Execute on desktop

Processor in the Loop (PIL)
- Numerical equivalence, model to target code
- Execute on target board

Hardware in the Loop (HIL)
- Check real-time behavior of the design and code.
- Execute on Speedgoat target computer using Simulink Real-Time
IDNEO Accelerates Development of AUTOSAR Software Components and Complex Device Drivers with Model-Based Design

Challenge
Reduce development time for embedded software for automotive applications

Solution
Use MATLAB and Simulink to model AUTOSAR software components and complex device drivers, run simulation-based tests, and generate embedded C code

Results
- Development time cut by at least 50%
- 80% of errors detected before hardware testing
- Test harnesses and MISRA-compliant C code generated from models

“By using Model-Based Design for all AUTOSAR projects, we have cut development time by at least 50 percent while increasing the number of defects identified early in the design phase and reducing the number of defects found in hardware tests and beyond.”
- Joan Albesa, IDNEO

Link to article
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
- Embedded Coder
- Simulink Real-Time

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

MathWorks Training Services: mathworks.com/training-schedule/simulation-based-testing-with-Simulink

Visit our Demo Booth