Growing Complexity of Embedded Systems

McKendrick, J. “Cars become ‘datacenters on wheels’, carmakers become software companies,” ZDJNet, 2013
Why do 71% of Embedded Projects Fail?

Poor Requirements Management

Sources: Christopher Lindquist, Fixing the Requirements Mess, CIO Magazine, Nov 2005
Key Takeaways

- Author, manage requirements in Simulink
- Early verification to find defects sooner
- Automate manual verification tasks
- Workflow that conforms to safety standards

"Reduce costs and project risk through early verification, shorten time to market on a certified system, and deliver high-quality production code that was first-time right"  Michael Schwarz, ITK Engineering

MATLAB EXPO 2018
Lear Delivers Quality Body Control Electronics Faster Using Model-Based Design

Challenge
Design, verify, and implement high-quality automotive body control electronics

Solution
Use Model-Based Design to enable early and continuous verification via simulation, SIL, and HIL testing

Results
- Requirements validated early. Over 95% of issues fixed before implementation, versus 30% previously
- Development time cut by 40%. 700,000 lines of code generated and test cases reused throughout the development cycle
- Zero warranty issues reported

“We adopted Model-Based Design not only to deliver better-quality systems faster, but because we believe it is a smart choice. Recently we won a project that several of our competitors declined to bid on because of its tight time constraints. Using Model-Based Design, we met the original delivery date with no problem.”
- Jason Bauman, Lear Corporation

[Link to user story]
Model Based Design Verification Workflow

1. Develop functions, perform ad-hoc testing, implement traceability
2. Refine design, Validate and Verify
3. Automatically detect quality issues and run-time error
4. Generate Code & Deploy
5. Auto-execute functional tests, verify product vs specification & Auto-report

Simulink Models

Requirements → Executable Specification → Model used for production code generation → Generated code

Equivalence checking

Component and system testing

Review and static analysis

Equivalence testing
Challenges with Requirements

Where are requirements implemented?

Is design and requirements consistent?

How are they tested?

Requirements

Executable Specification

Model used for production code generation

Simulink Models

C/C++

Generated code
Gap Between Requirements and Design

Requirements

Executable Specification

Model used for production code generation

Simulink Models

C/C++

Generated code
Simulink Requirements

Author

Track

Manage

If the Cancel switch is pressed, the value of reqDrv should be set to reqMode.Cancel.
Requirements Editor

To create a new requirement set to store requirements, click New Requirement Set. Save the requirement set to assign a name.

To add a requirement to a requirement set, select the requirement set and click Add Requirement. In the Properties pane, enter details for the requirement.

To add a child requirement, right-click a requirement and select Add Child Requirement.

To link a requirement to a block in your model, select the block, then right-click the requirement and select Link from "object name" (object type). A link appears in the Links pane.

For information on linking using the Requirements Perspective, see Getting Started in the documentation.

To view a list of links, select Links from the View dropdown list in the toolstrip.

Change the source - destination relationship by selecting a link, and choosing a Type from the dropdown list in the Properties pane.
To create a new requirement set to store requirements, click **New Requirement Set**. Save the requirement set to assign a name.

To add a requirement to a requirement set, select the requirement set and click **Add Requirement**. In the **Properties** pane, enter details for the requirement.

To add a child requirement, right-click a requirement and select **Add Child Requirement**.

To link a requirement to a block in your model, select the block, then right-click the requirement and select **Link from "object name" (object type)**. A link appears in the **Links** pane.

For information on linking using the Requirements Perspective, see **Getting Started** in the documentation.

To view a list of links, select **Links** from the **View** dropdown list in the toolbar.

Change the source - destination relationship by selecting a link, and choosing a **Type** from the dropdown list in the **Properties** pane.
Requirements Perspective
Requirement Reuse across Projects

Project Models

Shared Models

Shared Requirements

Project A

Project B

Links

Project Reqs

Shared Links

Project Reqs
Import Requirements from External Sources

- Microsoft Word
- IBM Rational DOORS
- ReqIF
- Simulink Requirements Editor

**3.1 Enabling cruise control**

Cruise control is enabled when the following conditions are met:

- Vehicle speed is within the target speed range (40km/h – 100km/h).
- Key position is ON.
- Gear position is Drive.
- Cruise button is pushed while the cruise control mode is disabled.
Requirements Import with ReqIF Standard

Allows you to work with requirements from third party tools in Simulink

- Import requirements from third party tools using ReqIF standard (Requirements Interexchange Format)
- Import wizard supports mapping custom attributes
- Tools that support ReqIF standard:
  - IBM DOORS / DOORS Next Generation
  - Siemens Polarion
  - PTC Integrity
REQ 3.1 ENABLING CRUISE CONTROL

Cruise control is enabled when …..

IMPLEMENTED BY

ENABLE SWITCH DETECTION

If the Enable switch is pressed …..

VERIFIED BY

Test Case

MATLAB EXPO 2018
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>Failed</td>
</tr>
<tr>
<td>2</td>
<td>#19</td>
<td>Cruise Control Mode</td>
<td>Implemented</td>
<td>Passed</td>
</tr>
<tr>
<td>2.1</td>
<td>#20</td>
<td>Disable Cruise Control system</td>
<td>Implemented</td>
<td>Passed</td>
</tr>
<tr>
<td>2.2</td>
<td>#24</td>
<td>Operation mode determination</td>
<td>Implemented</td>
<td>Passed</td>
</tr>
</tbody>
</table>

**Implementation Status**
- Implemented
- Justified
- Missing

**Verification Status**
- Passed
- Failed
- No Result
- Missing
Respond to Change

Original Requirement
If the switch is pressed and the counter reaches 50 then it shall be recognized as a long press of the switch.

Updated Requirement
If the switch is pressed and the counter reaches 75 then it shall be recognized as a long press of the switch.
Functional Testing

Does the design meet requirements?

Is it functioning correctly?

Is it completely tested?

Requirements → Executable Specification → Model used for production code generation → C/C++ → Generated code

Simulink Models

Does the design meet requirements?

Is it functioning correctly?

Is it completely tested?
Systematic Functional Testing

Test Case

Inputs
- MAT file (input)
- Signal Builder
  - Group 1
  - Signal 1

Test Sequence
- and more!

Assessments
- MAT file (baseline)
- MATLAB Unit Test
  - Test Assessment
    - and more!

MATLAB EXPO 2018
Model Coverage Analysis to Measure Testing

- Identify testing gaps
- Missing requirements
- Unintended Functionality
- Design Errors

Coverage Reports
Prove That Design Meets Requirements

- Prove design properties using formal requirement models
- Model functional and safety requirements
- Generates counter example for analysis and debugging

Requirements → Executable Specification → Model used for production code generation → Generated code

Simulink Models
Model Based Design Verification Workflow

1. Develop functions, perform ad-hoc testing, implement traceability

2. Refine design, Validate and Verify

3. Automatically detect quality issues and run-time error
Verify Design to Guidelines and Standards

- Is the design built right?
- Is it too complex?
- Is it ready for code generation?

Review and static analysis

Simulink Models

Requirements → Executable Specification → Model used for production code generation → C/C++ → Generated code
Automate verification with static analysis

Check for:
- Readability and Semantics
- Performance and Efficiency
- Clones
- And more......
Generate reports for reviews and documentation
Navigate to Problematic Blocks

<table>
<thead>
<tr>
<th>Block</th>
<th>Block Type</th>
<th>Code generation support</th>
<th>Recommendation for C/C++ production code deployment</th>
</tr>
</thead>
<tbody>
<tr>
<td>.../Intake Manifold/p0 = 0.589 bar</td>
<td>Integrator</td>
<td>Yes 1, 2</td>
<td>No</td>
</tr>
<tr>
<td>sldemo_fuelsys/Throttle Command</td>
<td>Repeating table</td>
<td>Yes 3</td>
<td>No</td>
</tr>
</tbody>
</table>

**Simulink Models**

- Requirements
- Executable Specification
- Model used for production code generation
- C/C++
- Generated code

RT/VM

p0 = 0.589 bar

0.41328

1/s

2 (rad/s)

N (rad/sec)
Recommended Action
Although Embedded Coder supports these blocks, they are not recommended for C/C++ production code deployment. Review the support notes for these blocks and follow the given advice.
Built in checks for industry standards and guidelines

- DO-178/DO-331
- ISO 26262
- IEC 61508
- IEC 62304
- EN 50128
- MISRA C:2012
- CERT C, CWE, ISO/IEC TS 17961
- MAAB (MathWorks Automotive Advisory Board)
- JMAAB (Japan MATLAB Automotive Advisory Board)
Configure and customize analysis

Simulink Models

- Requirements
- Executable Specification
- Model used for production code generation

C/C++

Generated code
Checks for standards and guidelines are often performed late

- Static Analysis
  - Requirements
  - Executable Specification
  - Simulink Models
  - Model used for production code generation
  - C/C++
  - Generated code
  - Rework
Shift Verification Earlier With Edit-Time Checking

- Highlight violations as you edit
- Fix issues earlier
- Avoid rework

Simulink Models

Requirements ➔ Executable Specification ➔ Model used for production code generation ➔ C/C++ ➔ Generated code

Edit-Time Checking

Highlight violations as you edit
Fix issues earlier
Avoid rework
Find Compliance Issues as you Edit with Edit-Time Checking
Assess Quality with Metrics Dashboard

- Consolidated view of metrics
  - Size
  - Compliance
  - Complexity

- Identify where problem areas may be
Grid Visualization for Metrics

- Visualize Standards
- Check Compliance
  - Find Issues
  - Identify patterns
  - See hot spots

Legend:
- Red: Fail
- Orange: Warning
- Green: Pass
- Gray: Not run
Detect Design Errors with Formal Methods

- Find run-time design errors:
  - Integer overflow
  - Dead Logic
  - Division by zero
  - Array out-of-bounds
  - Range violations

- Generate counter example to reproduce error
Model Based Design Verification Workflow

1. Develop functions, perform ad-hoc testing, implement traceability
2. Refine design, Validate and Verify
3. Automatically detect quality issues and run-time error
4. Generate Code & Deploy
5. Auto-execute functional tests, verify product vs specification & Auto-report
Equivalence Testing

Is the code functionally equivalent to model?

Is all the code tested?

Simulink Models

Requirements → Executable Specification → Model used for production code generation → Generated code → C/C++
Equivalence Testing

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

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

- Re-deploy model based tests on source compiled or compiled object
- Collect code coverage
Manage Testing and Test Results
Source Code Coverage Measurement & Comparison

- Identify testing gaps
- Missing requirements
- Unintended Functionality
- Design Errors
Qualify tools with IEC Certification Kit and DO Qualification Kit

- Qualify code generation and verification products
- Includes documentation, test cases and procedures

KOSTAL Asia R&D Center Receives ISO 26262 ASIL D Certification for Automotive Software Developed with Model-Based Design

BAE Systems Delivers DO-178B Level A Flight Software on Schedule with Model-Based Design
Learn More

Visit MathWorks Verification, Validation and Test Solution Page: mathworks.com/solutions/verification-validation.html
Summary

1. Author and manage requirements within Simulink
2. Find defects earlier
3. Automate manual verification tasks
4. Reference workflow that conforms to safety standards

Existing Workflow

- Component and system testing
- Review and static analysis
- Equivalence testing
- Equivalence checking

Simulink Models

- Requirements
- Executable Specification
- Model used for production code generation
- C/C++
- Generated code