MATLAB EXPO 2019

Comment obtenir des crédits de certification avec Simulink

Daniel Martins
Standards landscape

DO-178C
DO-254

IEC 61508

ISO 26262

IEC 62304

EN 50128
Development process for safety-related systems

- SW safety req. spec
- SW architecture
- SW system design
- Module design
- Coding
- Integration
- Validation testing
- Integration testing (components, subsystems)
- Integration testing (modules)
- Module testing

Partitioning

Compliance demonstration

System certification

Compliance demonstration

CERTIFIED
Requirement phase is key in the development process

Relative cost to fix an error

Errors introduced in:
- **coding phase**
- **design phase**
- **requirements phase**

**Project phase where error is fixed**

- **Requirements phase**: 1
- **Design phase**: 5-10 x
- **Coding phase**: 10-20 x
- **Testing phase**: 20-80 x

Model-Based Design: work early on requirements

- Model multidomain systems
- Explore and optimize system behavior
- Collaborate across teams and continents

- Generate efficient code
- Explore and optimize implementation tradeoffs
- Model concurrent systems

- Automate testing
- Detect design errors
- Support certification and standards
Role of Model-Based Design within DO-178C

A Design Model prescribes software component internal data structures, data flow, and/or control flow. A Design Model includes low-level requirements and/or architecture. In particular, when a model expresses software design data, regardless of other content, it should be classified as a Design Model. This includes models used to produce code.
Role of Model-Based Design within ISO 26262

A model consists of function blocks with well-defined inputs and outputs. [...] The functional model can serve as a blueprint for the implementation of embedded software on the control unit through code generation. [...] In comparison to code-based software development with a clear separation of phases, in model-based development a stronger coalescence of the phases “Software safety requirements”, “Software architectural design” and “Software unit design and implementation” can be noted [...]. Verification activities can also be treated differently since models can be used as a useful source of information for the testing process (e.g. model-based testing), or can serve as the object to be verified. The seamless utilization of models facilitates highly consistent and efficient development.
# Meeting DO-178C Objectives Table A5

<table>
<thead>
<tr>
<th>Objective</th>
<th>Software Levels</th>
<th>Anticipated Certification Credit [Tool(s)]</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) Source Code complies with low-level requirements</td>
<td>A, B, C</td>
<td>Full [Simulink Code Inspector]</td>
</tr>
<tr>
<td>(2) Source Code complies with software architecture</td>
<td>A, B, C</td>
<td>Full [Simulink Code Inspector]</td>
</tr>
<tr>
<td>(3) Source Code is verifiable</td>
<td>A, B</td>
<td>Full [Simulink Code Inspector + Polyspace Bug Finder]</td>
</tr>
<tr>
<td>(4) Source Code conforms to standards</td>
<td>A, B, C</td>
<td>Full [Polyspace Bug Finder]</td>
</tr>
<tr>
<td>(5) Source Code is traceable to low-level requirements</td>
<td>A, B, C</td>
<td>Full [Simulink Code Inspector]</td>
</tr>
<tr>
<td>(6) Source Code is accurate and consistent</td>
<td>A, B, C</td>
<td>Partial [Simulink Code Inspector, Polyspace verifier]</td>
</tr>
</tbody>
</table>
How does Simulink Code Inspector work?

IR: Intermediate Representation

Simulink Model .slx/.mdl → Embedded Coder → C/C++ source code .c/.cpp

Model IR

Code IR

IR transformations

Normalized Model IR

Normalized Code IR

Matching

Simulink Code Inspector

Code Inspection Report .html

Error and status messages in command line or GUI
Simulink Code Inspector Report for GearControl.slx

Overall Inspection Result: Passed
Utils Need Manual Review: No

Code Verification Results: Verified

Function Interface Verification Results: Verified

<table>
<thead>
<tr>
<th>Function</th>
<th>Status</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>GearControl_initialize</td>
<td>Verified</td>
<td></td>
</tr>
<tr>
<td>GearControl</td>
<td>Verified</td>
<td></td>
</tr>
</tbody>
</table>

Model To Code Verification Results: Verified

<table>
<thead>
<tr>
<th>Status</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Verified</td>
<td>Model objects with status Verified: 18</td>
</tr>
<tr>
<td></td>
<td>Model objects with status Partially processed: 0</td>
</tr>
<tr>
<td></td>
<td>Model objects with status Unable to process: 0</td>
</tr>
<tr>
<td></td>
<td>Model objects with status Failed to verify: 0</td>
</tr>
</tbody>
</table>

Code To Model Traceability Results: Traced

<table>
<thead>
<tr>
<th>Status</th>
<th>Number of model objects</th>
</tr>
</thead>
<tbody>
<tr>
<td>Traced</td>
<td>18</td>
</tr>
<tr>
<td>Partially processed</td>
<td>0</td>
</tr>
<tr>
<td>Unable to process</td>
<td>0</td>
</tr>
<tr>
<td>Failed to trace</td>
<td>0</td>
</tr>
</tbody>
</table>

Not processed code:

File: GearControl.c

<table>
<thead>
<tr>
<th>Code location</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>#include &quot;GearControl.h&quot;</td>
</tr>
<tr>
<td>17</td>
<td>#include &quot;GearControl_private.h&quot;</td>
</tr>
</tbody>
</table>
Meeting IS026262 Objectives

- Table 7: Methods for Software Unit Verification

<table>
<thead>
<tr>
<th>Methods</th>
<th>ASIL</th>
<th>MBD Tools</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1n Back-to-back comparison test between model and code, if</td>
<td>++</td>
<td>Simulink Test</td>
</tr>
<tr>
<td>applicable</td>
<td></td>
<td>Embedded Coder SIL/PIL</td>
</tr>
</tbody>
</table>

- Table 9: Structural Coverage Metrics at the Software Unit Level

<table>
<thead>
<tr>
<th>Methods</th>
<th>ASIL</th>
<th>MBD Tools</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1a Statement coverage</td>
<td>++</td>
<td>Simulink Coverage</td>
</tr>
<tr>
<td>1b Branch coverage</td>
<td>+</td>
<td>Simulink Coverage</td>
</tr>
<tr>
<td>1c MC/DC Modified (Condition/Decision Coverage)</td>
<td>+</td>
<td>Simulink Coverage</td>
</tr>
</tbody>
</table>
Back-to-back testing
Simulink Coverage report

Summary

<table>
<thead>
<tr>
<th>File Contents/Complexity</th>
<th>Decision</th>
<th>Statement</th>
<th>Function</th>
<th>Function call</th>
</tr>
</thead>
<tbody>
<tr>
<td>ShiftThreshold.c</td>
<td>8 100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>ShiftThreshold</td>
<td>7 100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>ShiftThreshold_initialize</td>
<td>1 100%</td>
<td>100%</td>
<td>100%</td>
<td>-</td>
</tr>
</tbody>
</table>

Details

1. File **ShiftThreshold.c**

   Function: **ShiftThreshold** (line 21)
   **ShiftThreshold_initialize** (line 84)

   Metric | Coverage
   --- | ---
   Cyclomatic Complexity | 8
   Decision (D1) | 100% (8/8) decision outcomes
   Statement | 100% (20/20) covered statements
   Function | 100% (2/2) covered functions
   Function call | 100% (6/6) covered function calls

2. Function **ShiftThreshold** (line 21)

   File: **ShiftThreshold.c**
   Model Object: **ShiftThreshold**
   Covered expressions: **int32_t *rtu_gear** (line 30)
   **int32_t *rtu_gear** (line 59)

   Metric | Coverage
   --- | ---
   Execution coverage | 100%
What DO-178C says about tool qualification

Qualification of a tool is needed when processes of this document are eliminated, reduced, or automated by the use of a software tool without its output being verified.

The purpose of the tool qualification process is to ensure that the tool provides confidence at least equivalent to that of the process(es) eliminated, reduced, or automated.
What ISO26262-8 says about tool qualification

A software tool used in the development of a system or its software or hardware elements, can support or enable a tailoring of the safety-lifecycle [...]. In such cases confidence is needed that the software tool effectively achieves the following goals:

– the risk of systematic faults in the developed product due to malfunctions of the software tool leading to erroneous outputs is minimized, and

– the development process is adequate with respect to compliance with ISO 26262, if activities or tasks required by ISO 26262 rely on the correct functioning of the software tool used.
## DO-178C Tool Classification

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Tools that</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>could insert an error.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>could fail to detect an error, and are used to eliminate/reduce:</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1. Other verification process(es)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2. Development process(es) impacting the software</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>could fail to detect an error.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Software Level</th>
<th>Criteria</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td>A</td>
<td>TQL-1</td>
</tr>
<tr>
<td>B</td>
<td>TQL-2</td>
</tr>
<tr>
<td>C</td>
<td>TQL-3</td>
</tr>
<tr>
<td>D</td>
<td>TQL-4</td>
</tr>
</tbody>
</table>
DO-178C Tool qualification methods

<table>
<thead>
<tr>
<th>TQL5</th>
<th>TQL4</th>
<th>TQL3, TQL2, TQL1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tool Operational Requirements</td>
<td>Tool Requirements</td>
<td>Tool Development Plan</td>
</tr>
<tr>
<td>Tool Operational V&amp;V Cases &amp; Procedures</td>
<td>Tool Qualification Plan</td>
<td>Tool Verification Plan</td>
</tr>
</tbody>
</table>
MathWorks DO Qualification Kit (for DO-178)

- Tool Operational Requirements
- Test cases and procedures
- Tool Qualification plan

- Polyspace Bug Finder
- Polyspace Code Prover
- Simulink Requirements
- Simulink Report Generator
- Simulink Check
- Simulink Coverage
- Simulink Code Inspector
- Simulink Test
- Simulink Design Verifier
- Model Comparison
ISO26262 Tool classification

I. Tool Classification

- Tool Use Cases
  - UC 1…n
- Tool Impact Analysis
  - TI 1
- Tool Error Detection
  - TD 1
  - TD 2
  - TD 3
- Tool Confidence Level
  - TCL 1
  - TCL 2
  - TCL 3

II. Tool Qualification

Source: ISO 26262-8, 11.4.4.2, Tables 3 & 4

- Qualification measures for TCL 3 based on ASIL
- Qualification measures for TCL 2 based on ASIL
- No further qualification measures required

Integration of the tool into the Model-Based Design workflow
Possibility that the tool can introduce errors or fail to detect them
Confidence in error prevention and detection measures
Classification of the software tool into confidence levels

Tool Qualification Measures
## ISO26262 Tool qualification methods

<table>
<thead>
<tr>
<th>Methods</th>
<th>ASIL</th>
<th>MBD Tools</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>1a Increased confidence from use in accordance with 11.4.7</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td>1b Evaluation of the tool development process in accordance to 11.4.8</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td>1c Validation of the software too in accordance with 11.4.9</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>1d Development in accordance with a safety standard</td>
<td>+</td>
<td>+</td>
</tr>
</tbody>
</table>
MathWorks IEC Certification Kit *(for ISO 26262 and IEC 61508)*

- Workflow description
- Tool Qualification plan
- TÜV SÜD Certificate
- Test cases and procedures

- Embedded Coder
- Simulink PLC Coder
- Polyspace Bug Finder
- Polyspace Code Prover
- Simulink Check
- Simulink Coverage
- Simulink Test
- Simulink Design Verifier
Summary

- Models are accepted by Standards
- Standards recognize benefits of tools
- Several Standards activities can be automated by models and tools
- MathWorks Certification/Qualification Kits describe those activities