Model-Based Design for ISO26262

Young Joon Lee
Principal Application Engineer
Agenda

- System Certification and Compliance Demonstration
- MathWorks Solution for ISO 26262
  - Pre-qualification of tools
  - Reference V&V workflow and integrated tool chain
- Case Study for V&V Workflow
  - V&V at Model Level
  - V&V at Code Level
- Conclusion & Questions
Certification, Standards, and Compliance Demonstration
ISO 26262 “Road Vehicles - Functional Safety”

- **Functional safety standard** for passenger cars
  - Concerned with avoidance of unreasonable risks due to hazards caused by malfunctioning E/E systems

- Facilitates modern software engineering concepts such as
  - Modeling and simulation
  - Early verification / validation
  - Code generation
ISO 26262 Standard

- 10 parts
- 400+ pages
- 100+ work products

ISO 26262-1
• Vocabulary
ISO 26262-2
• Management of functional safety
ISO 26262-3
• Concept phase
ISO 26262-4
• Product development: system level
ISO 26262-5
• Product development: hardware level
ISO 26262-6
• Product development: software level
ISO 26262-7
• Production and operation
ISO 26262-8
• Supporting processes
ISO 26262-9
• ASIL-oriented and safety-oriented analyses
ISO 26262-10
• Guideline

Back-to-back comparison test
Qualification of required resources
Software architectural design specification
Control flow analysis

Design and coding guidelines
Simulation of dynamic parts of the design

15+ software-related method tables
70+ methods

Reference phase model for the software development
ISO 26262 and Model-Based Design

- Model-Based Design is **deeply rooted in ISO 26262**

<table>
<thead>
<tr>
<th>1. Vocabulary</th>
</tr>
</thead>
<tbody>
<tr>
<td>2. Management of functional safety</td>
</tr>
<tr>
<td>3. Concept phase</td>
</tr>
<tr>
<td>4. Product development: system level</td>
</tr>
<tr>
<td>5. Product development: hardware level</td>
</tr>
<tr>
<td>6. Product development: software level</td>
</tr>
<tr>
<td>7. Production and operation</td>
</tr>
<tr>
<td>8. Supporting processes</td>
</tr>
</tbody>
</table>

Sections concerned with Model-Based Design
ISO 26262 and Model-Based Design

- Model-Based Design, early verification & validation, and code generation are **integral parts of ISO 26262** (Examples from part 1)

### ISO 26262-1

#### Table 2 — Notations for software architectural design

<table>
<thead>
<tr>
<th>Methods</th>
<th>ASIL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
</tr>
<tr>
<td>1a Informal notations</td>
<td>++</td>
</tr>
<tr>
<td>1b Semi-formal notations</td>
<td>+</td>
</tr>
<tr>
<td>1c Formal notations</td>
<td>+</td>
</tr>
</tbody>
</table>

#### Table 7 — Notations for software unit design

<table>
<thead>
<tr>
<th>Methods</th>
<th>ASIL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
</tr>
<tr>
<td>1a Natural language</td>
<td>++</td>
</tr>
<tr>
<td>1b Informal notations</td>
<td>++</td>
</tr>
<tr>
<td>1c Semi-formal notations</td>
<td>+</td>
</tr>
<tr>
<td>1d Formal notations</td>
<td>+</td>
</tr>
</tbody>
</table>
ISO 26262 and Model-Based Design

- Model-Based Design is **deeply rooted in ISO 26262**

1.74 model-based development

Development that uses models to describe the functional behavior of the elements which are to be developed.

NOTE or both. Depending on the level of abstraction used for such a model it can be used for simulation or code generation.

Annex B
(informative)

Model-based development

B.1 Objectives

This Annex describes the **concept of model-based development of in-vehicle software** and outlines its implications on the product development at the software level.

**The seamless utilization of models facilitates a highly consistent and efficient development.**
ISO 26262 and Model-Based Design

- ISO 26262-6 enables code generation and early verification and validation

**NOTE 4** For model-based development, software unit testing can be carried out at the model level followed by back-to-back tests between the model and the code. The back-to-back tests are used to ensure that the behaviour of the models with regard to the test objectives is equivalent to the automatically-generated code.

**Table 14 — Structural coverage metrics at the software unit level**

<table>
<thead>
<tr>
<th>Methods</th>
<th>ASIL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
</tr>
<tr>
<td>1a Statement coverage</td>
<td>++</td>
</tr>
<tr>
<td>1b Branch coverage</td>
<td>+</td>
</tr>
<tr>
<td>1c MC/DC (Modified Condition/Decision Coverage)</td>
<td>+</td>
</tr>
</tbody>
</table>

**NOTE 2** In the case of model-based development, software unit testing may be moved to the model level using analogous structural coverage metrics for models.
Development Process for High-Integrity Applications: System Certification

- Recognition by a certification authority (e.g. TÜV) that an in-vehicle system complies with the requirements of a standard
Development Process for High-Integrity Applications: Compliance Demonstration

- System certification involves a compliance demonstration:
  - applicant must provide evidence that the objectives of the standard were met

- Compliance demonstration is a lengthy and labour-intensive process
Using the ISO 26262 Reference Model to Construct a Visual Gap Analysis Framework

### 6. Product Development at SW Level

<table>
<thead>
<tr>
<th>Process Steps</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>6-5</td>
<td>Initiation of product development</td>
</tr>
<tr>
<td>6-7</td>
<td>Software architectural design</td>
</tr>
<tr>
<td>6-8</td>
<td>Software unit design &amp; imp</td>
</tr>
<tr>
<td>6-9</td>
<td>Software unit testing</td>
</tr>
<tr>
<td>6-10</td>
<td>Software integration &amp; testing</td>
</tr>
<tr>
<td>6-11</td>
<td>Verification of software safety req</td>
</tr>
</tbody>
</table>

#### Methods

Table 1-16

![Diagram of process steps and methods]
Using the ISO 26262 Reference Model to Construct a Visual Gap Analysis Framework

6. Product Development at SW Level

- 6-5 Initiation of product development
- 6-7 Software architectural design
- 6-8 Software unit design & imp
- 6-9 Software unit testing
- 6-10 Software integration & testing
- 6-11 Verification of software safety req

Table 1 — Topics to be covered by modelling and coding guidelines

<table>
<thead>
<tr>
<th>Topic</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>1a</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td>1b</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td>1c</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td>1d</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td>1e</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td>1f</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td>1g</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td>1h</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td>1i</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
</tbody>
</table>

Methods

Process Steps
Using the ISO 26262 Reference Model to Construct a Visual Gap Analysis Framework

6. Product Development at SW Level

6-5 Initiation of product development
6-7 Software architectural design
6-8 Software unit design & imp
6-9 Software unit testing
6-10 Software integration & testing
6-11 Verification of software safety req
Using the ISO 26262 Reference Model to Construct a Visual Gap Analysis Framework

6. Product Development at SW Level

6-5 Initiation of product development
6-7 Software architectural design
6-8 Software unit design & imp
6-9 Software unit testing
6-10 Software integration & testing
6-11 Verification of software safety req
Traditional Development for ISO 26262 ASIL D Safety Functions

- **Observations**
  - It is difficult to sufficiently cover some of the tables with methods.
    - e.g. table 6 – Simulation of dynamic parts of design is highly recommended
    - Furthermore, additional methods will significantly improve quality and reduce second guessing justification of not performing certain practices
  - While traditional tooling provides automation, there is still a lot of manual effort going on here
**Model-Based Design for ISO 26262 ASIL D Safety Functions**

- **Observations**
  - Many of the advanced analysis and design techniques called out by the standard are manually intensive to perform using traditional methods
    - e.g. Range checks of input/output data, Diverse SW Design, Prototype generation
  - Model-Based Design supports many of methods called out by the standard and provides automation to further reduce the manual effort

### Process Steps

<table>
<thead>
<tr>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>lj</td>
</tr>
<tr>
<td>li</td>
</tr>
<tr>
<td>lh</td>
</tr>
<tr>
<td>lg</td>
</tr>
<tr>
<td>lf</td>
</tr>
<tr>
<td>le</td>
</tr>
<tr>
<td>ld</td>
</tr>
<tr>
<td>lc</td>
</tr>
<tr>
<td>lb</td>
</tr>
<tr>
<td>ia</td>
</tr>
</tbody>
</table>

|       | Table 1 | Table 2 | Table 3 | Table 4 | Table 5 | Table 6 | Table 7 | Table 8 | Table 9 | Table 10 | Table 11 | Table 12 | Table 13 | Table 14 | Table 15 | Table 16 |
|-------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|
|       | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      |
|       | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      |
|       | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      |
|       | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      |
|       | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      |
|       | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      |
|       | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      |
|       | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      |
|       | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      |
|       | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      |
|       | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      |
|       | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      |
|       | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      |
|       | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      |
|       | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      |
|       | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      |
|       | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      |
|       | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      | ++      |
Adopting Capabilities to Optimize Model-Based Design for ISO 26262

Manual Effort

Design Capabilities & Quality

Time
MathWorks Solution for ISO26262
MathWorks Solution for ISO26262

- Pre-qualification of tools
- Reference workflow and integrated tool chain
  - Supporting MBD & early verification
  - Modeling guidelines
  - Traceability (Requirements – Model – Code)
  - Equivalence testing
ISO 26262-8 Qualification of Software Tools

Objective

The first objective of this clause is to provide criteria to determine the required level of confidence in a software tool when applicable.

The second objective of this clause is to provide means for the qualification of the software tool when applicable, in order to create evidence that the software tool is suitable to be used to tailor the activities or tasks required by ISO 26262

cf. ISO 26262-8, 11.1
Tool Qualification Approach

- Tool classification must be carried out and documented for all tools.

- For tools classified at TCL 2 or higher, at least one of the following qualification methods shall be applied and documented:
  a) Increased confidence from use
  b) Evaluation of the tool development process
  c) Validation of the software tool
  d) Development in compliance with a safety standard

- Tools classified at TCL1 need no qualification measures.
# Tool Qualification Methods

- **Selection criteria:**
  - Suitability for all ASILs
  - Combination of 2+ different methods to ensure robustness w.r.t. expected future changes to ISO 26262

<table>
<thead>
<tr>
<th>Method</th>
<th>TCL 2</th>
<th>TCL 3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ASIL A</td>
<td>ASIL B</td>
</tr>
<tr>
<td>1a Increased confidence from use</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td>1b Evaluation of the tool development process</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td>1c Validation of the software tool</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>1d Development in compliance with a safety standard</td>
<td>+</td>
<td>+</td>
</tr>
</tbody>
</table>

Note: Embedded Coder, Simulink Verification and Validation, Simulink Design Verifier, and Polyspace products for C/C++ were not developed using certified processes.
Tool Qualification Work Products

**Tool Qualification Planning**
- **Software Tool Qualification Plan (STQP)**
  - Applicant, application information (incl. max. ASIL)
  - Tool name, tool version, tool configuration, operational environment
  - Tool use case(s)
  - Available means to detect malfunctions or erroneous output of the tool.

**Tool Documentation**
- **Software Tool Documentation (STD)**
  - Tool overview
  - Available tool documentation set
  - Operational environment and constraints
  - Installation instructions
  - Known issues

**Tool Classification**
- **Software Tool Classification Analysis (STCA)**
  - Tool error detection
  - Tool confidence level
  - Tool qualification methods

**Tool Qualification**
- **Software Tool Qualification Report (STQR)**
  - Evidence that the tool qualification has been carried out as planned
  - Usage constraints and malfunctions identified during the qualification (if any)
Tool Qualification for COTS Tools

Workshare

Tool qualification process may involve multiple parties:

- **Tool user**
  - Responsible for *final tool qualification* in the context of the application

- **MathWorks**
  - Carries out *generic pre-qualification* based on reference workflow
  - Supports / streamlines user’s activities by providing an *ISO 26262 tool qualification kit*

- **Certification Authority**
  - Provides an *independent assessment* of reference workflow and tool qualification kit
  - Issues an certificate
Assessment Results for Embedded Coder

Assessment report
Tool Qualification Workshare

I. Pre-qualification
   A. Generic tool classification [MathWorks]
   B. Generic pre-qualification [MathWorks]
   C. Independent assessment [Cert. authority]

ISO 26262 tool qualification kit
   - Generic work products (pre-filled templates)
   - Assessment results (assessment report, certificate)

II. Application-specific adaptation
   A. Review / adaptation of the tool qualification kit [Tool user]

ISO 26262 tool qualification work products
   - Final work products (completed templates)
   - Assessment results (assessment report, certificate)
IEC Certification Kit
Includes ISO 26262 Tool Qualification Kit

Supports engineers who use MathWorks™ products to develop, verify, or validate software for systems that must comply with, or be certified according to ISO 26262

- **Support for tool qualification**
  - Pre-filled templates for ISO 26262-8 tool qualification work products
  - Evidence of independent assessment (certificates, assessment reports)
  - Tool for managing qualification artifacts (Certification Artifacts Explorer)

- **Support for certification-related software development activities**
  - Reference Workflow with Conformance Demonstration Template
  - Utility functions (Traceability matrix generation)

The IEC Certification Kit product also provides support for the IEC 61508 base standard and other application specific standards
The IEC Certification Kit product allows users to re-use and adapt the pre-qualification results for their ISO 26262 projects.

Evidence for independent assessment:
- Assessment report
- Certificate

Templates for tool qualification work products

Reference workflow with conformance demonstration template

www.mathworks.com/products/iec-61508/
MathWorks Solution for ISO26262

- Pre-qualification of tools
- **Reference workflow and integrated tool chain**
  - Supporting MBD & early verification
  - Modeling guidelines
  - Traceability (Requirements – Model – Code)
  - Equivalence testing
ISO26262 Reference Workflow for Verification and Validation and Code Generation
Coverage of ISO 26262–6 and -8 Requirements

**Table 7 – Notations for software unit design**

<table>
<thead>
<tr>
<th>Methods</th>
<th>ASIL</th>
<th>Applicable Model-Based Design Tools / Processes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1a Natural language</td>
<td>A++</td>
<td>• Simulink - Model Info block, DocBlock</td>
</tr>
<tr>
<td></td>
<td>B++</td>
<td>• Simulink Verification and Validation – System Requirements block, Requirements Management Interface (RMI)</td>
</tr>
<tr>
<td>1b Informal notations</td>
<td>A++</td>
<td></td>
</tr>
<tr>
<td></td>
<td>B++</td>
<td></td>
</tr>
<tr>
<td></td>
<td>C+</td>
<td>• Simulink Report Generator – Web View, System Design</td>
</tr>
</tbody>
</table>

**Table 9 – Methods for the verification of the software unit design and implementation (1/2)**

<table>
<thead>
<tr>
<th>Methods</th>
<th>ASIL</th>
<th>Applicable Model-Based Design Tools / Processes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1a Walk-through</td>
<td>A++</td>
<td>• Simulink</td>
</tr>
<tr>
<td>1b Inspection</td>
<td>A++</td>
<td>• Simulink Report Generator – Web View, System Design</td>
</tr>
<tr>
<td></td>
<td>B+</td>
<td>• Simulink Design Verifier – Test case generation</td>
</tr>
<tr>
<td></td>
<td>C+</td>
<td>• Simulink – Stateflow</td>
</tr>
<tr>
<td></td>
<td>D+</td>
<td>• Embedded Coder – Processor-in-the-Loop (PiL) testing, Code metrics report</td>
</tr>
</tbody>
</table>

**Table 10 – Methods for software unit testing (2/2)**

<table>
<thead>
<tr>
<th>Methods</th>
<th>ASIL</th>
<th>Applicable Model-Based Design Tools / Processes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1b Interface test</td>
<td>A++</td>
<td>• Simulink</td>
</tr>
<tr>
<td>1c Fault injection test</td>
<td>B+</td>
<td>• Simulink – Stateflow</td>
</tr>
<tr>
<td>1d Resource usage test</td>
<td>C+</td>
<td>• Embedded Coder – Processor-in-the-Loop (PiL) testing, Code metrics report</td>
</tr>
<tr>
<td>1e Back-to-back test between model and code, if applicable</td>
<td>D+</td>
<td>• Simulink Verificaction and Validation – Component testing capabilities, Model coverage</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Simulink Design Verifier – Test case generation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Embedded Coder – Software-in-the-Loop (SiL) testing, Processor-in-the-Loop (PiL) testing, code generation Verification (CGV)</td>
</tr>
</tbody>
</table>

www.mathworks.com/automotive/standards/iso-26262.html
V&V Workflow for ISO 26262
V&V Workflow for ISO 26262 with MathWorks Products

- Model Advisor, Modeling standards checking
- Simulation (model testing), Model coverage, RMI
- Module and integration testing at the model level
- Review and static analysis at the model level
- PIL testing using Embedded IDE Links
- Embedded Coder traceability report
  Or
  Model vs. code coverage comparison
- Equivalence testing
- Prevention of unintended functionality
- Model used for production code generation
- Generated C code
- Object code
- Textual requirements
- Executable specification
- Modeling
- Simulink / Stateflow / Simulink Fixed Point
- Embedded Coder
- Code generation
- Compilation and linking
Defining the System
Model Verification
Example ISO 26262 Workflow for Model-Based Design with MathWorks Products

Textual requirements → Executable specification → Model used for production code generation

Review and static analysis at the model level

Module and integration testing at the model level

Equivalence testing

Prevention of unintended functionality

Modeling

Code generation

Compilation and linking

Generated C code → Object code

Simulink / Stateflow / Simulink Fixed Point
Requirement Specification

TCU (Simple)
Software Functional Requirements Document
The MathWorks Company

Functional Requirements

TCU (Simple)
Software Test Requirements Document
The MathWorks Company

Testing Requirements
Modeling with Simulink

- Create executable specifications
- Design with simulation

Advantages:
- Eliminate ambiguity
- Facilitate team communication and component-based modeling
- Analyze and improve requirements and design through simulation
- Test system performance before building physical prototypes
- Automate document generation
Simple TCU Algorithm Model

TCU (Simple)

<table>
<thead>
<tr>
<th>Module Type</th>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCU</td>
<td>Function</td>
<td>...</td>
</tr>
</tbody>
</table>

Functional Requirements

Simple TCU Model
Test Cases to Signal Builder

1. TC01 - Passing Maneuver
   This test case tests with 10% throttle value to check passing maneuver and steady gear number increase.

2. TC02 - Gradual Acceleration
   This test case tests gear change behavior with gradual speed increase with steady increase of throttle pedal position.

Test Cases

Signal Builder
Module Test-Harness

Test Cases (Signal Builder)

Model (Model Block)

Output Check (Assertion Block)
Example ISO 26262 Workflow for Model-Based Design with MathWorks Products

Simulation (model testing), Model coverage, RMI

Module and integration testing at the model level

Review and static analysis at the model level

Equivalence testing

Prevention of unintended functionality

Textual requirements → Executable specification → ... → Model used for production code generation → Generated C code → Object code

Modeling

Simulink / Stateflow / Simulink Fixed Point

Code generation

Compilation and linking
Creating links between textual documents and model objects
Requirements Traceability – Report
Simulink Verification and Validation

- Requirements Report provides screenshots of the model and lists all the associated requirements.
Traceability Matrix Generation

Generate traceability information covering requirements, model elements, and generated code

- Use MS Excel to review, track, and annotate generated traceability matrices for your project
- Fulfill ISO 26262 requirements to document traceability information and to demonstrate absence of unintended functionality

(cf. ISO 26262-6, 8.4.6)

![Traceability Matrix Generation](image)

Requires: ‘IEC Certification Kit’ for IEC 61508 and ISO 26262;
‘Embedded Coder’
Example ISO 26262 Workflow for Model-Based Design with MathWorks Products

Model Advisor, Modeling standards checking

Simulation (model testing), Model coverage, RMI

Module and integration testing at the model level

Review and static analysis at the model level

Equivalence testing

Prevention of unintended functionality

Textual requirements → Executable specification → Model used for production code generation → Generated C code → Object code

Modeling

Simulink / Stateflow / Simulink Fixed Point

Code generation

Compilation and linking
Modeling Standards Checking Overview
Simulink Verification and Validation

- Static analysis of models against a set of checks
  - Checks for simulation
  - Checks for code generation
  - Requirements Consistency
  - Modeling Standards

- Modeling Standards Checks for:
  - MAAB Style Guidelines
  - DO-178B
  - IEC 61508
  - ISO 26262

- Extensibility API
MAAB Style Guidelines

- MathWorks™ Automotive Advisory Board (MAAB)

- Consistency
- Interoperability
- Error prevention
- Knowledge sharing
Leverage industry-best practices and MathWorks tool expertise when developing high-integrity systems

- Modeling Guidelines and corresponding Model Advisor checks to facilitate modeling standards and guidelines objectives of ISO 26262, IEC 61508, DO-178B, and MISRA-C

ISO 26262 Model Checks
Example ISO 26262 Workflow for Model-Based Design with MathWorks Products
Module Test-Harness

Test Cases (Signal Builder)  Model (Model Block)  Output Check (Assertion Block)
Coverage metrics identifies untested portions of your model

Coverage Report for sbr

Test 1
Started Execution: 27-Feb-2008 13:36:21
Ended Execution: 27-Feb-2008 13:36:21

Summary

<table>
<thead>
<tr>
<th>Model Hierarchy/Complexity</th>
<th>Test 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>C1</td>
</tr>
<tr>
<td>1. sbr</td>
<td>50 %</td>
</tr>
<tr>
<td>1. . . Inputs</td>
<td>3 50%</td>
</tr>
<tr>
<td>2. . . . . Automatic Tests</td>
<td>2 50%</td>
</tr>
<tr>
<td>4. . . Outputs Assertions</td>
<td>2 100%</td>
</tr>
<tr>
<td>5. . . . Verification Subsystem</td>
<td>2 100%</td>
</tr>
<tr>
<td>6. . . SRR_Logic</td>
<td>22 74%</td>
</tr>
<tr>
<td>7. . . . SRR</td>
<td>22 74%</td>
</tr>
<tr>
<td>8. . . . SF_SRR</td>
<td>22 74%</td>
</tr>
<tr>
<td>9. . . . . SF_SFF_ON</td>
<td>15 75%</td>
</tr>
<tr>
<td>10. . . SF_SFF_UNFASTEN</td>
<td>8 75%</td>
</tr>
<tr>
<td>11. . . SF_HIGH_SPEED</td>
<td>4 75%</td>
</tr>
<tr>
<td>12. . . SRR_Logical</td>
<td>22 5%</td>
</tr>
<tr>
<td>13. . . . SRR</td>
<td>22 5%</td>
</tr>
</tbody>
</table>

Test Results:
- Substate executed 100%
  - State "HIGH_SPEED": 89/187/106
  - State "LOW_SPEED": 97/8/187/106

Substate exited when parent exits 50%
  - State "HIGH_SPEED": 0/1
  - State "LOW_SPEED": 1/1
Improving Test Suite
Simulink Design Verifier

- Generating tests to reach coverage criteria

Test generation harness with the copy of the original model

Test inputs that ensure complete coverage
Automated Documentation – Report Generation

- Use model to handle documentation information
  - DocBlock
  - ModellInfo

- Generate documentation from model
  - Custom reports using Simulink Report Generator

- System Design Description
Code Verification
Example ISO 26262 Workflow for Model-Based Design with MathWorks Products

Model Advisor, Modeling standards checking

Simulation (model testing), Model coverage, RMI

Module and integration testing at the model level

Review and static analysis at the model level

Model used for production code generation

Generated C code

Object code

Textual requirements → Executable specification → …

Modeling

Equivalence testing

Prevention of unintended functionality

Embedded Coder

Simulink / Stateflow / Simulink Fixed Point

Code generation

Compilation and linking
Code Generation

- Use of Legacy Code Tool for introduction of existing C code on Simulink models
- ISO ANSI-C production code generation with Embedded Coder.
- Possible prototyping on target
- Run-time errors verification with PolySpace before the compilation phase and execution on the target
Most Development is on Core Software Algorithms
Interfacing Generated Code

- Share entry points, data and parameters with header files
- Call generated model functions from OS/Scheduler
A diagram illustrating an example ISO 26262 workflow for model-based design with MathWorks products. The workflow involves several steps starting from textual requirements and ending with object code. Key tools and processes include:

- **Model Advisor** for checking standards
- **Simulation (model testing)**
- **Model coverage, RMI**
- **Module and integration testing at the model level**
- **Review and static analysis at the model level**
- **Equivalence testing**
- **Prevention of unintended functionality**
- **Model used for production code generation**
- **Generated C code**
- **Embedded Coder traceability report** or **Model vs. code coverage comparison**
- **Simulink / Stateflow / Simulink Fixed Point**
- **Modeling**
- **Code generation**
- **Compilation and linking**

The workflow diagram shows the flow from requirements to production code, highlighting key steps and tools used in the process.
Traceability

- **Tracing Requirements ↔ Model**  
  Simulink® Verification and Validation™

- **Tracing Model ↔ Source Code**  
  Embedded Coder™

- **Tracing Requirements ↔ Source Code**  
  Simulink Verification and Validation
Example of EC HTML Report

Hyperlink within C code files.

Hyperlink back to model.

Hyperlink from model to code.
Tracing Requirements ↔ Source Code
Simulink Verification and Validation
Embedded Coder

- Including requirements in the generated source code
Example ISO 26262 Workflow for Model-Based Design with MathWorks Products

Model Advisor, Modeling standards checking

Simulation (model testing), Model coverage, RMI

Module and integration testing at the model level

Review and static analysis at the model level

Embedded Coder traceability report or Model vs. code coverage comparison

Equivalence testing

Prevention of unintended functionality

Polyspace for code verification

Textual requirements → Executable specification → Model used for production code generation → Generated C code → Object code

Modeling

Simulink / Stateflow / Simulink Fixed Point

Code generation

Compilation and linking

Embedded Coder
Runtime Errors

- **Arithmetic errors**
  - Found in (code)
    - Overflows, division by zero, bit-shifts, square root of negative numbers
  - Caused by (model)
    - Faulty scaling, changes in or unknown calibrations, untested data ranges coming out of a subsystem into an arithmetic block

- **Memory corruption**
  - Found in (code)
    - Out of bound array indexes
    - Pointer arithmetic
  - Caused by (model)
    - Array manipulation in Stateflow
    - Hand-written look-up table functions

- **Data truncation**
  - Found in (code)
    - Overflows
    - Wrap around
  - Caused by (model)
    - Saturations leading to unexpected data flow inside the generated code

- **Coding errors**
  - Found in (code)
    - Non initialized data
    - Dead code leading to unreachable transitions or states
  - Caused by (model)
    - Faulty Stateflow programming
Polyspace products for code verification

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

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

- **Process**
  - Use early in development
  - For automatically generated and handwritten code
Tracing results back to the model
PolySpace™ Model Link SL

- Integrates code verification with the production code generation
Proving Code Correctness
PolySpace Server for C/C++

1. Authoring
   - Configuration - selection of code base and types of problems to focus on

2. Execution and Reporting
   - Runtime benchmark for both generated and legacy code

Benefits
- Detects hard-to-find runtime problems
- Increases confidence by proving absence of runtime errors
- Helps with independent verification for certification purposes
Example ISO 26262 Workflow for Model-Based Design with MathWorks Products

- **Model Advisor, Modeling standards checking**
- **Simulation (model testing), Model coverage, RMI**
- **Module and integration testing at the model level**
- **Review and static analysis at the model level**

**PIL testing using Embedded IDE Links**

- **Embedded Coder traceability report**
- **Model vs. code coverage comparison**
- **Equivalence testing**
- **Prevention of unintended functionality**
- **Polyspace for code verification**

**Artifact Flows**

- **Textual requirements**
- **Executable specification**
- **Model used for production code generation**
- **Generated C code**
- **Object code**
- **Code generation**
- **Compilation and linking**

**Tools and Components**

- **Simulink / Stateflow / Simulink Fixed Point**
- **Modeling**
- **Embedded Coder**
Processor-in-the-Loop Testing:
Verify Production Controller with Processor-in-the-loop

Execution
- Host/Target
- Nonreal-time
Summary of V&V Workflow for ISO 26262 with MathWorks Products

- Simulation (model testing), Model coverage, RMI
- Module and integration testing at the model level
- Review and static analysis at the model level
- Model Advisor, Modeling standards checking
- PIL testing using Embedded IDE Links
- Equivalent testing
- Prevention of unintended functionality
- Model used for production code generation
- Generated C code
- Object code

Textual requirements → Executable specification → Simulink / Stateflow / Simulink Fixed Point → Modeling → Code generation → Embedded Coder → Compilation and linking