HOW TO ENSURE ISO 26262 FOR HIGHLY COMPLEX SOFTWARE MODELS

Dr. Heiko Doerr

QUALITY IN THE DRIVER’S SEAT

SOLUTIONS FOR INTEGRATED QUALITY ASSURANCE OF EMBEDDED AUTOMOTIVE SOFTWARE
Integration of ISO-26262-6 SW process into model-based development

Model testing as dedicated process steps

Architectural complexity

Handling of model quality

Summary and conclusion

Disclaimer: Examples are using the Simulink / Stateflow approach to model-based SW development in the Automotive domain. Results can be transferred to other modeling languages and paradigms.
Model-based development has become the dominant approach to SW development in the automotive domain.

- Functionality is realized as a **model**
- Large models consist of **more than 5,000 blocks**
- **Generation of production code** from the model
- ISO 26262 recommends model-based development and requires a number of quality assurance measures
Models contribute to various tasks during development. Model-based SW development focusses on the generation of code.
The ISO 26262-6 reference model schedules validation at several phases. Model-based development offers frontloading and reuse of test artifacts.
ALLOCATION OF ISO-26262-6 PHASES

SW-safety activities are covered by model-based development process.

Requirements specification

- Requirements
- Behavioral model
- (RCP) Code

System and function design

- System integration
  - 6-11 Verif. of SW safety reqs

6-6 Safety requirements

6-7 Architectural design

6-8 Unit design & Implementation

6-9 Unit testing

6-10 Integration and testing

6-11 Verif. of SW safety reqs

Implementation model

Automatic code generation

(Target) Code

Implement.
The standard highly recommends requirements driven tests. Automated generation of test cases from models is not sufficient.

Table 10 — Methods for software unit testing

<table>
<thead>
<tr>
<th>Methods</th>
<th>ASIL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
</tr>
<tr>
<td>1a Requirements-based test$^a$</td>
<td>++</td>
</tr>
<tr>
<td>1b Interface test</td>
<td>++</td>
</tr>
<tr>
<td>1c Fault injection test$^b$</td>
<td>+</td>
</tr>
<tr>
<td>1d Resource usage test$^c$</td>
<td>+</td>
</tr>
<tr>
<td>1e Back-to-back comparison test between model and code, if applicable$^d$</td>
<td>+</td>
</tr>
</tbody>
</table>

Table 11 — Methods for deriving test cases for software unit testing

<table>
<thead>
<tr>
<th>Methods</th>
<th>ASIL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
</tr>
<tr>
<td>1a Analysis of requirements</td>
<td>++</td>
</tr>
<tr>
<td>1b Generation and analysis of equivalence classes$^a$</td>
<td>+</td>
</tr>
<tr>
<td>1c Analysis of boundary values$^b$</td>
<td>+</td>
</tr>
<tr>
<td>1d Error guessing$^c$</td>
<td>+</td>
</tr>
</tbody>
</table>
Requirements driven tests continuously assure safety requirements. Comprehensive model testing reduces overall efforts by frontloading.
WHAT IS THE MES TEST MANAGER?

A professional model test management framework which supports requirements-based unit testing of Simulink® models.
TEST DATA DEFINITION

- Multiple ways to define test sequences
- Import (measured) data as test data
  - Support of different file formats (mat, csv, Excel, ... files)
  - Import via command line or GUI
- Systematic test definition (requirements based)
  - Classification tree method (equivalence class method)
  - Integration of CTE/XL Standard/Professional
    - Graphical test specification
- Formal test specification (requirements based)
  - Model Test Case Designer (MTCD) integrated into MTest
    - Script-based test specification
- Further (internal) specification methods
  - TTCN derivation
// Example – left curve

[+1s] // wait time
  speed = ramp(50);
[+3s] // acceleration
[+2s] // constant drive
  steering angle = sin(120);
  speed = ramp(25);
[+3s] // left steering
[+2s] // drive in curve
  steering angle = sin(0);
  speed = ramp(50);
[+3s] // steering back
[+1s] // wait time
TEST CASE DEFINITION WITH CTE

Properties
- Standard: TCSpecification | TestSequenceProperty
- Name: TS_BZ_0001 Alarm-Warnsignal und Rangfolge der Eingangssignale
- ID = 48

Description
- Text:
  <b>Anforderung</b>
  RQ_BZ_0001, RQ_BZ_0002, RQ_BZ_0004
  <b>Testablauf</b>

Diagram:
- Buzzed
  - to_Start
  - to_Stop
  - nu16_MaxCntOn
  - nu16_MaxCntOn

Table:
- TS_BZ_0001 Alarm-Warnsignal und Rangfolge der Eingangssignale
  - Start
  - Buzzed_Start aktiviert
  - Buzzed_Stop wird aktiviert
  - Buzzed_Start wird zurückgesetzt
  - Buzzed_Start wieder aktiviert
  - Buzzed_Start zurückgesetzt
  - Buzzed_Start und Buzzed_Stop zurückgesetzt
  - Beide auf 1 setzen
  - Beide zurücksetzen
  - Buzzed_Start aktiviert
  - Buzzed_Stop aktivieren
  - Buzzed_Start und Buzzed_Stop werden zurückgesetzt
  - Ende

- TS_BZ_0002 Pulsweilmodulation
  - Anfang
  - Buzzed_Start aktivieren
  - Buzzed_Stop aktivieren
  - zurücksetzen
  - Buzzed_Start aktivieren
  - Buzzed_Stop aktivieren
  - zurücksetzen
  - Buzzed_Start aktivieren
  - Buzzed_Stop aktivieren
  - zurücksetzen
  - Ende
TEST EXECUTION – AUTOMATIC SIMULATION

- Test execution process
  - Loading test setup
  - Loading simulation data (inputs / test description)
  - Setting simulation parameters
  - Executing test bed with data (>> sim(…))
  - Saving simulation results (outputs)

- Support for different simulation modes
  - Model simulation
  - Code simulation
  - etc.
TEST ASSESSMENTS

- Test assessments
  - Evaluation of requirements based on input/output signals
  - Assessments: implemented as m-files

- Types of assessments
  - Global assessment: applicable to all test sequences
  - Local assessment: applicable to one test sequences

- Automatic evaluation
  - One result for each assessment per test sequence
    - Passed: Requirement passed
    - Failed: Requirement failed
    - Not Applicable: Req. not triggered
  - Aggregation for each test sequence
  - Most critical result is sequence result
- Assessment function acts as a check or reference to a software requirement
- I/O variables are evaluated by the assessment for each test sequence

Diagram:

- Requirement
  - Check
  - In-Signals
  - Out-Signals
  - Local-Signals
  - Parameter

Result: m Assessments

n Test Sequence

Result:

Model Engineering Solutions Gmbh | How to Ensure ISO 26262 | May 2015
EXAMPLE: AUTOMATED TEST ASSESSMENT

Test and assessment
for requirement
REQ:TVC_001
“Absolute tank volume
should never be higher than
80 liters (MaxVolume)”
Model testing requires a high degree of automation to cope with large models.

- Reduce manual tasks to set-up, test case specification and evaluation
- Use back-to-back test to maintain safety requirements
- Simplify test evaluation by automated assessments validating safety invariants
Implementation of modelling (and coding) guidelines is “STRONGLY RECOMMENDED”.

<table>
<thead>
<tr>
<th>Topics</th>
<th>ASIL</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>1a</strong> Enforcement of low complexity(^a)</td>
<td>++</td>
</tr>
<tr>
<td><strong>1b</strong> Use of language subsets(^b)</td>
<td>++</td>
</tr>
<tr>
<td><strong>1c</strong> Enforcement of strong typing(^c)</td>
<td>++</td>
</tr>
<tr>
<td><strong>1d</strong> Use of defensive implementation techniques</td>
<td>++</td>
</tr>
<tr>
<td><strong>1e</strong> Use of established design principles</td>
<td>++</td>
</tr>
<tr>
<td><strong>1f</strong> Use of unambiguous graphical representation</td>
<td>++</td>
</tr>
<tr>
<td><strong>1g</strong> Use of style guides</td>
<td>++</td>
</tr>
<tr>
<td><strong>1h</strong> Use of naming conventions</td>
<td>++</td>
</tr>
</tbody>
</table>

\(^{a}\) This topic is recommended for lower complexity projects.

\(^{b}\) Use of language subsets is important for maintaining consistency and understanding.

\(^{c}\) Strong typing helps in reducing errors related to type mismatches.
Model architecture analysis identifies complex model parts easily.
SW-UNITS AS ATOMIC ARCHITECTURAL ENTITY

Definition of a SW unit is essential for safety activities, because …

- SW units are the atomic entity for
  - development
  - quality assurance
  - delivery

- Early definition of terms is essential for definition of development process

- Early definition is essential for architectural design

- ISO 26262 Part 6:

  8.4.3 The specification of the software units shall describe the functional behaviour and the internal design to the level of detail necessary for their implementation.
Definition of a SW unit faces conflicting interests.
The SW architecture will be populated by subsystems from models.
SW units will be represented by selected subsystems.

(INGREDIENTS OF SW ARCHITECTURE)

SW Architect.

Model Structure

SW Component

SW Unit

Subsystem
ARCHITECTURE ANALYSIS FOR LARGE MODELS

Architecture analysis is essential to control complexity in large models.

- Clear visualization of model architecture and complexity
- Delivers realistic figures on model size
- Better estimates for the allocation of resources for development, testing, and review of models
- Detection of unbalanced functionality and models
- Guidance to refactoring and creation of libraries
- Detection of model clones
MODELING GUIDELINES

Modeling guidelines capture best practices, contribute to model quality and implementation of safety-related software.

Guidelines for …

- unique look and feel
  support readability and better detection of errors in the model

- allowed sub-sets of modeling language
  prevent from the use of risky features

- consistent configuration of tools and models
  avoid common and known errors

- code generator and compiler settings
  prevent from unintended use of generation options
Readability is a pre-requisite for safety.
Strong data typing is by intention not supported by Simulink/Stateflow. Specific checks required to assure the requested property.

\[ \text{Int16, } 2^{-17}, [-0.25..0.24] \]

\[ \text{UInt16, } 2^{-10}, [0..63.999] \]

\[ \text{Inherited, SAT:[0..63]} \]

\[ \text{UInt16, } 2^{-7}, [0..511.9] \]
Automation and organizational support is required to efficiently maintain coherent requirements on model quality.
A comprehensive set of automatic quality measures contribute to the implementation of ISO 26262-6 in particular for large SW-models.
CONCLUSION

Model-based SW development can comply with safety requirements. Moreover, it helps to compensate additional safety efforts.

- Increase of productivity via code generation
  - Safe up to 50% of implementation time

- Significant improvement of software quality
  - Up to 40% less software errors

- Reduction of development time and costs
  - Safe up to 30%
MODEL ENGINEERING SOLUTIONS GMBH

Mauerstrasse 79
10117 Berlin

T: +49 30 2091 6463-0
F: +49 30 2091 6463-33

info@model-engineers.com
www.model-engineers.com