THE CHALLENGE OF ISO 26262 FOR COMPLEX SOFTWARE MODELS

Oliver Collmann

QUALITY IN THE DRIVER’S SEAT

SOLUTIONS FOR INTEGRATED QUALITY ASSURANCE OF EMBEDDED AUTOMOTIVE SOFTWARE
OUR COMPANY

□ WHO WE ARE

- Founded in 2006 in Berlin out of Mercedes-Benz Research
- A staff of 50+ experts in the field of engineering, testing and functional safety
- Clear focus on model-based engineering and quality assurance for MBD

□ WHAT WE DO

- Integrated process support for all stages of model-based software development
- We develop software tools for quality assurance of model-based software

SOFTWARE PUTS YOU ON THE ROAD.
INTEGRATED PROCESS SUPPORT FOR MBD

- MANAGEMENT SUPPORT
- QUALITY ENGINEERING
- QUALITY TOOLS
- TRAINING WORKSHOPS

INTEGRATED APPROACH FOR MBD
AGENDA

- ISO 26262 and 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 modelling languages and paradigms.
Model-based development dominates SW development in Automotive

- Functionality is realized as a **model**
- **Generation of production code** from the model
- Enabler for early-stage **quality assurance**
- Makes **distributed engineering** more efficient

Traditional software development  Model-based software development
Models contribute to various tasks during development. Model-based SW development focusses on the generation of code.
The ISO 26262 reference model schedules validation at several phases. Model-based development offers frontloading and reuse of test artifacts.
SW-safety activities are covered by model-based development process.

Requirements specification

System and function design

Requirements

System integration

6-11 Verif. of SW safety reqs

6-10 Integration and testing

SW integration

6-6 Safety requirements

6-7 Architectural design

6-8 Unit design & Implementation

6-9 Unit testing

Automatic code generation

Implement.

Implementation model

Behavioral model

Physical model

(RCP) Code

(Target) Code
Automotive software models are often very complex
- Many levels 7 +
- Many blocks 8000 +
- Complex signalling

Too complex to test or walk through manually

Example MES “Huge Model”
AGENDA

- ISO 26262 and 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 modelling languages and paradigms.
The standard highly recommends requirement-based tests. Automated random generation of test cases is not sufficient.

### Table 10 — Methods for software unit testing

<table>
<thead>
<tr>
<th>Methods</th>
<th>ASIL</th>
</tr>
</thead>
<tbody>
<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>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>
SOFTWARE INTEGRATION AND (UNIT) TESTING

Requirements driven tests continuously assure safety requirements. Comprehensive model testing reduces overall efforts by frontloading.

SW safety Requirements

Test input data

Model

Model-in-the-Loop

C Code (Host)

Software-in-the-Loop

C Code (Target)

Processor-in-the-Loop
Model testing requires a high degree of automation to cope with large models.

- Test execution platform: reduce manual tasks to set-up, test case specification
- Back-to-back test to maintain safety requirements
- Automated assessments to validate test results
AGENDA

- ISO 26262 and model-based development
- Model testing as dedicated process steps
- Architectural complexity
- Handling of model quality
- Summary and conclusion
ENFORCEMENT OF LOW COMPLEXITY “STRONGLY RECOMMENDED”. 

Table 3 — Principles 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 Hierarchical structure of software components</td>
<td>++</td>
</tr>
<tr>
<td>1b Restricted size of software components</td>
<td>++</td>
</tr>
<tr>
<td>1c Restricted size of interfaces</td>
<td>+</td>
</tr>
<tr>
<td>1d High cohesion within each software component</td>
<td>+</td>
</tr>
<tr>
<td>1e Restricted coupling between software components</td>
<td>+</td>
</tr>
<tr>
<td>1f Appropriate scheduling properties</td>
<td>++</td>
</tr>
<tr>
<td>1g Restricted use of interrupts</td>
<td>+</td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>Notes</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>In methods 1b, 1c, 1e and 1g &quot;restricted&quot; means to minimize in balance with other design considerations.</td>
</tr>
<tr>
<td>b</td>
<td>Methods 1d and 1e can, for example, be achieved by separation of concerns which refers to the ability to identify, encapsulate, and manipulate those parts of software that are relevant to a particular concept, goal, task, or purpose.</td>
</tr>
<tr>
<td>c</td>
<td>Method 1e addresses the limitation of the external coupling of software components.</td>
</tr>
<tr>
<td>d</td>
<td>Any interrupts used have to be priority-based.</td>
</tr>
</tbody>
</table>
Model architecture analysis identifies complex model parts easily.
Architecture analysis is essential to control complexity in large models.

- Clear visualization of model architecture and complexity
- Realistic indicators on model and component size
- Better allocation of resources for development, testing, and review of models
- Detection of clones and unbalanced functionality
- Guidance to refactoring and creation of libraries
AGENDA

- ISO 26262 and model-based development
- Model testing as dedicated process steps
- Architectural complexity
- Modelling guidelines
- Summary and conclusion
STAGED DETERMINATION OF GUIDELINES

Browse through modeling guidelines

State-of-the-art is defined by approx. 700+ individual guidelines

- Mathworks Automotive Advisory Board
- MISRA Autocode TargetLink
- MISRA Autocode Simulink / Stateflow
- dSPACE TargetLink guidelines
- MES Functional Safety Guidelines
STAGED DETERMINATION OF GUIDELINES

Reduce redundancy and resolve inconsistencies by expert assessment.

- Assessment of guidelines to filter out redundancy and inconsistencies
- Impact of the individual guidelines must be fully understood before removing a specific guideline
- Consolidation revealed that nearly 35% of the guidelines are overlapping
STAGED DETERMINATION OF GUIDELINES

Identify the most appropriate phase for application of guideline.

- ISO 26262-6 generally requests application to “Software unit design and implementation”
- Model-based software development allows frontloading and early application of guidelines
- Each guideline is scheduled to the appropriate phase of development
SPECIFIC EXAMPLE OF A ISO 26262 GUIDELINE

Strong data typing is by intention not supported by Simulink/Stateflow. Specific checks are required to assure the requested property.

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

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

Inherited, SAT: $[0 .. 63]$

UInt16, $2^{-7}$, $[0 .. 511.9]$
AGENDA

- ISO 26262 and model-based development
- Model testing as dedicated process steps
- Architectural complexity
- Model Quality and modelling guidelines
- Summary and conclusion
- Automate testing and test evaluation
- Automate static review and guideline management
- Manage model complexity and redundancy
- Refine architecture and restructure model into coherent components
- Use library and model references in a consistent way
CONCLUSION

MBD helps to meet safety requirements.

- 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
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