ISO 26262 Conform Model Based Development and Verification Process

dSPACE User Conference India 2010

Adrian Valea
BTC Embedded Systems AG
 Agenda

- Introduction
- Theoretical aspects of the ISO 26262 standard, its terminology, methodology and mapping
  - ISO 26262 – New Functional Safety Standard
  - Enhanced Model Based Development and Testing
  - Model-Based Reference Workflow
  - Modeling and Coding Guidelines
  - Formal Specifications and Formal Verification
  - Automatic Test Generation and Execution
  - Requirements Based Testing and Traceability
  - Qualification of software tools in the context of ISO 26262
- Conclusions
BTC Embedded Systems AG

- OSC GmbH Company established in 1999
- OSC – Embedded Systems AG founded in 2002
- Beginning 2009 OSC became BTC-ES
  - as part of BTC AG Corporation with 1400 Employees
- BTC-ES Headquarter in Oldenburg (D)
  - Subsidiary in Munich (D)
  - BTC Japan Co., Ltd.
- Expert in Automatic Test- and Validation Technologies

dSPACE Strategic Partner provider of Automatic Test and Verification Products for TargetLink
- Common Activities especially together with dSPACE GER/JP/FRA
ISO 26262 – New Functional Safety Standard

- New Automotive Standard addressing functional safety
- Derived from IEC 61508
- Draft International Standard (DIS) published in July 2009
- Official release planned for 2011
- But already used by OEMs and suppliers
ISO 26262 – Automotive Safety Integrity Levels

- ISO 26262 defines four Automotive Safety Integrity Levels (ASIL)
- Definition of ASIL: one class to specify the necessary safety requirements items for achieving an acceptable residual risk with D representing the highest and A the lowest class.

<table>
<thead>
<tr>
<th>ISO 26262</th>
<th>IEC 61508</th>
</tr>
</thead>
<tbody>
<tr>
<td>(normal QM)</td>
<td>-</td>
</tr>
<tr>
<td>ASIL A</td>
<td>SIL 1</td>
</tr>
<tr>
<td>ASIL B</td>
<td>SIL 2</td>
</tr>
<tr>
<td>ASIL C</td>
<td>SIL 3</td>
</tr>
<tr>
<td>ASIL D</td>
<td>SIL 4</td>
</tr>
</tbody>
</table>
ISO 26262 – Model-based Development

ISO 26262 specifically addresses model-based development and testing

One characteristic of the model-based development paradigm is the fact that the functional model not only specifies the desired function but also provides design information and finally even serves as the basis of the implementation by means of code generation.

In contrast 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. Moreover, one and the same graphical modeling notation is used during the consecutive development stages. Testing activities are also treated differently since models can be used as a useful source of information for the testing process (model-based testing). The seamless utilization of models facilitates a highly consistent and efficient development.
Enhanced Model Based Development and Testing Process

- **Textual Requirements**
- **Specification Model**
- **Design Model**
- **Source Code**
- **Object Code**
- **Integration Component**
- **System Component**

- **Testing Phases**:
  - Formal Specification
  - Functional Tests
  - Structural Tests
  - Coverage Reports
  - Automatic Test Specification
  - Automatic Test Execution

- **Modeling**
- **Design**
- **Code Generation**
- **Build**
- **Formal Verification (FV)**
- **Software Integration**
- **System Integration**

BTC Embedded Systems AG proprietary · All rights reserved
Model-Based Reference Workflow

- Well suited to develop safety-related software according to ISO 26262 and IEC 61508

- Many of the proposed methods are directly recommended by ISO 26262 and IEC 61508

- TÜV Certification
  - Workflow has been approved by TÜV
  - TargetLink and EmbeddedTester are fit for purpose to develop safety-related software according to ISO DIS 26262, IEC 61508 and derivative standards such as EN 50128

1 EN 50128, standard for software for railway control and protection systems, is considered as a sector-specific standard derived from IEC 61508.
Model-Based Reference Workflow

Textual Requirements (High Level Req.) → Modeling → Model (Low Level Req.) → Code generation → Compile Link → Obj. Code

- Software Design Standard: Modeling guidelines and guideline checking
- Software Coding Standard: Coding guidelines and guideline checking
- Model verification (MIL simulation, Formal Verification Model Review)
- Back-to-back testing (MIL vs. PIL simulation)

Testing methods:
- Requirements based testing
- Structural testing for coverage analysis
- Performance testing
Modeling and Coding Guidelines

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

<table>
<thead>
<tr>
<th>Topics</th>
<th>ASIL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
</tr>
<tr>
<td>1a Enforcement of low complexity</td>
<td>++</td>
</tr>
<tr>
<td>1b Use of language subsets</td>
<td>++</td>
</tr>
<tr>
<td>1c Enforcement of strong typing</td>
<td>++</td>
</tr>
<tr>
<td>1d Use of defensive implementation techniques</td>
<td>0</td>
</tr>
<tr>
<td>1e Use of established design principles</td>
<td>+</td>
</tr>
<tr>
<td>1f Use of unambiguous graphical representation</td>
<td>+</td>
</tr>
<tr>
<td>1g Use of style guides</td>
<td>+</td>
</tr>
<tr>
<td>1h Use of naming conventions</td>
<td>++</td>
</tr>
</tbody>
</table>
Requirements Traceability

- Requirements in DOORS Excel Word, etc. can be linked to the model

- Links: Model ↔ Code

- Bi-Directional Traceability between Requirements and Test-Cases
Formal Specifications and Formal Verification in the context of ISO 26262
Formal Specification and Formal Verification Workflow

- Formal Specification
- Embedded Validator®
- Formal Verification
- Model Checking
- Textual Requirements
- Modeling
- TargetLink Model
- Formal Requirement
- Embedded Validator®

BTC Embedded Systems AG proprietary · All rights reserved
## ISO 26262 - Notation Recommendations

### Table 3 — 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 8 — 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 Documentation of the software unit design in 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>

- Formal Notations are recommended for all Design levels starting with ASIL A
**ISO 26262 – Formal Verification Recommendations**

**Semi-formal Verification** (→ Simulation) of Requirements is even highly recommended for levels greater than ASIL B

**Formal Verification** recommended from ASIL B

⇒ inline with Model-based Development

- Executable Specification/Model allows Semi-formal verification
- Formal Verification becomes applicable in early Development stages
Automatic Test Generation and Execution in the context of ISO 26262
Automatic Test Generation and Execution Workflow

TargetLink

Model

Source Code

Test Vectors

EmbeddedTester
Model Coverage Measurement

EmbeddedTester
Code Coverage Measurement

Automatic Hierarchical Back-to-back testing (MIL vs. PIL)

EmbeddedTester
Automatic Hierarchical Test Vector Generation

Model

Source Code

Object Code

TargetLink

Compiler

BTC Embedded Systems AG proprietary · All rights reserved
### ISO 26262 - Back-to-Back Testing

For Testing of SW-Units, from ASIL C back-to-back-Tests are highly recommended
- Model-based and Code Testing in MIL, SIL and PIL

<table>
<thead>
<tr>
<th>Methods</th>
<th>ASIL A</th>
<th>ASIL B</th>
<th>ASIL C</th>
<th>ASIL D</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirement-based test</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td>Interface test</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td>Fault injection test</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>++</td>
</tr>
<tr>
<td>Resource usage test</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>++</td>
</tr>
<tr>
<td>Back-to-back test between model and code, if applicable</td>
<td>+</td>
<td>+</td>
<td>++</td>
<td>++</td>
</tr>
</tbody>
</table>

---

*a* This includes injection of arbitrary faults in order to test safety mechanisms (e.g. by corrupting values of variables)

*b* Some aspects of the resource usage test can only be evaluated properly when the software unit tests are executed on the target hardware or if the emulator for the target processor supports resource usage tests.

*c* This method requires a model that can simulate the functionality of the software units. Here, the model and code are stimulated in the same way and results compared with each other.
ISO 26262 - Coverage Metrics (SW-Unit)

9.4.4 To evaluate the completeness of test cases and to demonstrate that there is no unintended functionality, the coverage of requirements at the software unit level shall be determined and the structural coverage shall be measured in accordance with the metrics listed in Table 14. If necessary, additional test cases shall be specified or a rationale shall be available.

Table A — 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>

- Quality of Test Cases measured
  - ... by coverage of Requirements (just informally)
  - ... by structural Coverage metrics
    - The higher the ASIL-Level, the stronger the Metrics
  - Structural coverage metrics highly recommended for all ASIL Levels.
ISO 26262 - Target Testing

9.4.5 The test environment for software unit testing shall correspond as far as possible to the target environment. If the software unit testing is not carried out in the target environment, the differences in the source and object code, and the differences between the test environment and the target environment, shall be analysed in order to specify additional tests on the target environment during the subsequent test phases.

NOTE 2 Depending on the scope of the tests, it can be useful to carry out the software unit testing on the processor of the target system. If this is not possible, a processor emulator can be used. Otherwise the software unit testing is executed on the development system.

NOTE 3 Software unit testing can be executed in different environments, for example:

— Model-in-the-loop tests;
— Software-in-the-loop tests;
— Processor-in-the-loop tests; and

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.

- Perfect Match for model-based Development
- PIL-Tests are appropriate
Requirements-Based Testing

Requirements Based Testing and Traceability in the context of ISO26262
Requirements-Based Testing Workflow

- Textual Requirements
- Test Vectors
- Requirement Based Test Vector Creation
  - Manual or Tool-Based
- EmbeddedTester
  - Automatic Hierarchical Requirement based testing
  - Test Execution Platform

- MiL
- TargetLink
- Source Code
- SiL
- Compiler
- PIL
- Object Code
ISO 26262 - Requirements-based Test

Requirements-based Test is highly recommended for all ASIL–Levels (also Integration Testing)

Metrics for Quality of Tests just intuitively defined
Tools coverage of ISO 26262 standard methods

TargetLink and EmbeddedTester features mapping on ISO26262
Which portion of that workflow is covered by a tool?
## TargetLink Coverage of ISO26262 standard

<table>
<thead>
<tr>
<th>SW Dev. Subphase</th>
<th>ISO 26262-6 Reference</th>
<th>ASIL A</th>
<th>ASIL B</th>
<th>ASIL C</th>
<th>ASIL D</th>
<th>Model Level</th>
<th>Code Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initiation of product development at the software level</td>
<td>Table 1 – Topics to be covered by modelling and coding guidelines</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>MISRA AC TL guidelines, dSPACE TargetLink guidelines, MAAB guidelines</td>
<td>MISRA C:2004 guidelines, TargetLink MISRA C compliance document</td>
</tr>
<tr>
<td></td>
<td>1a Enforcement of low complexity</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Table 1 – 1b Use of language subsets</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Table 1 – 1c Enforcement of strong typing</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Table 1 – 1d Use of defensive implementation techniques</td>
<td>0</td>
<td>+</td>
<td>++</td>
<td>++</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Table 1 – 1e Use of established design principles</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>++</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Table 1 – 1f Use of unambiguous graphical representation</td>
<td>+</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Table 1 – 1g Use of style guides</td>
<td>+</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Table 1 – 1h Use of naming conventions</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Specification of software safety requirements</td>
<td>Table 2 – Methods for the verification of requirements</td>
<td>+</td>
<td>+</td>
<td>++</td>
<td>++</td>
<td>Supported by creating models that are executed in order to verify that it meets its requirements. Simulink/TargetLink MIL simulation</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1c Semi-formal verification</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Embedded Validator supports formal verification of TargetLink models</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1d Formal Verification</td>
<td>0</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

BTC Embedded Systems AG proprietary · All rights reserved
## TargetLink Coverage of ISO26262 standard

<table>
<thead>
<tr>
<th>SW Dev. Subphase</th>
<th>ISO 26262-6 Reference</th>
<th>ASIL A</th>
<th>ASIL B</th>
<th>ASIL C</th>
<th>ASIL D</th>
<th>Model Level</th>
<th>Code Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software architectural design</td>
<td>Table 3 – Notations for software architectural design 1b Semi-formal notations</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>- Using Simulink for the creation of a model addressing software architectural design aspects.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Table 4 – Principles for software architectural design 1a Hierarchical structure of software components</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>- Simulink allows modularization using hierarchical subsystems. - Simulink provides means to measure module complexity.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1b Restricted size of software components</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Table 7 – Methods for the verification of the software architectural design 1a Informal verification by walkthrough of the design</td>
<td>++</td>
<td>+</td>
<td>0</td>
<td>0</td>
<td>- Informal verification is used to assess whether the software requirements are completely and correctly refined and realised. Being able to link requirements and model and navigate back and forth supports this assessment and facilitates verifying that all requirements are covered.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Table 7 – 1b Informal verification by inspection of the design</td>
<td>+</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Table 7 – 1c Semi-formal verification by simulating dynamic parts of the design</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>- Simulink/TargetLink MIL simulation</td>
<td></td>
</tr>
</tbody>
</table>
## TargetLink Coverage of ISO26262 standard

<table>
<thead>
<tr>
<th>SW Dev. Subphase</th>
<th>ISO 26262-6 Reference</th>
<th>ASIL A</th>
<th>ASIL B</th>
<th>ASIL C</th>
<th>ASIL D</th>
<th>Model Level</th>
<th>Code Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software Unit Design and Imple-</td>
<td>Table 8 – Notations</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>- The implementation model serves as the software unit design specification</td>
<td></td>
</tr>
<tr>
<td>mentation</td>
<td>for software unit design</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1c Semi-formal notations</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Table 11</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>- Direct visual inspection of the model</td>
<td>- MISRA C compliance checkers to assure compliance to MISRA C</td>
</tr>
<tr>
<td></td>
<td>1c Model Inspection</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>- TargetLink report generation for modules together with implementation information like data types, scalings, and simulation plots</td>
<td>- TargetLink generated code in HTML-format with navigable links from code to model</td>
</tr>
<tr>
<td></td>
<td>Table 11 – 1d Model</td>
<td>++</td>
<td>+</td>
<td>0</td>
<td>0</td>
<td>- Rule-based guideline checkers to assure compliance with modelling guidelines</td>
<td>- Can be replaced by automated methods and techniques</td>
</tr>
<tr>
<td></td>
<td>Walkthrough</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Table 11 – 1e Inspection of the source code</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Table 11 – 1f Walkthrough of the source code</td>
<td>++</td>
<td>+</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Process Phase</td>
<td>ISO 26262 Reference</td>
<td>ISO 26262 Method</td>
<td>ASIL A</td>
<td>ASIL B</td>
<td>ASIL C</td>
<td>ASIL D</td>
<td>EmbeddedValidator Coverage</td>
</tr>
<tr>
<td>---------------</td>
<td>---------------------</td>
<td>-----------------</td>
<td>--------</td>
<td>--------</td>
<td>--------</td>
<td>--------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>Requirements Specification</td>
<td>Table 8 — Notations for software unit design</td>
<td>1d Formal notations for requirements specification</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>Formal specification of functional and safety requirements based on patterns</td>
</tr>
<tr>
<td></td>
<td>Table 3 — Notations for software architectural design</td>
<td>1c Formal notations for requirements specification</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>Formal specification of functional and safety requirements based on patterns</td>
</tr>
<tr>
<td>Requirements Verification</td>
<td>Table 2 — Methods for the verification of requirements</td>
<td>1c Semi-formal verification</td>
<td>+</td>
<td>+</td>
<td>++</td>
<td>++</td>
<td>Self-monitoring validity of the C-Observes from Patterns under MIL/SIL/PIL simulation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1d Formal verification</td>
<td>0</td>
<td>0</td>
<td>+</td>
<td>+</td>
<td>Formal verification based on model checking</td>
</tr>
<tr>
<td></td>
<td>Table 7 — Methods for the verification of the software architectural design</td>
<td>1d Semi-formal verification by simulating dynamic parts of the design</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>Self-monitoring validity of the C-Observes from Patterns under MIL/SIL/PIL simulation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1e Formal verification</td>
<td>0</td>
<td>0</td>
<td>+</td>
<td>+</td>
<td>Formal verification based on model checking</td>
</tr>
<tr>
<td></td>
<td>Table 10 — Methods for the verification of software unit design and implementation</td>
<td>1b Semi-formal verification</td>
<td>+</td>
<td>+</td>
<td>++</td>
<td>++</td>
<td>Self-monitoring validity of the C-Observes from Patterns under MIL/SIL/PIL simulation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1c Formal verification</td>
<td>0</td>
<td>0</td>
<td>+</td>
<td>+</td>
<td>Formal verification based on model checking</td>
</tr>
<tr>
<td>Software integration and testing</td>
<td>Table 15 — Methods for software integration testing</td>
<td>1a Requirements-based test</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>Import and Execution of Functional Tests from different formats e.g. CTE, EXCEL, Signal Builder. Requirements based test generation based on C-Observers Patterns coverage</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1e Back-to-back test between model and code</td>
<td>+</td>
<td>+</td>
<td>++</td>
<td>++</td>
<td>Automatic MIL/SIL/PIL regression test execution and results comparison</td>
</tr>
<tr>
<td>Software unit testing</td>
<td>Table 14 — Structural coverage metrics at the software unit level</td>
<td>1a Statement coverage</td>
<td>++</td>
<td>++</td>
<td>+</td>
<td>+</td>
<td>Part of the code coverage report</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1b Branch coverage</td>
<td>+</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>Part of the code coverage report</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1c MC/DC (Modified Condition/Decision Coverage)</td>
<td>+</td>
<td>+</td>
<td>++</td>
<td>++</td>
<td>Part of the code coverage report</td>
</tr>
<tr>
<td>Software integration and</td>
<td>Table 17 — Structural coverage metrics at the</td>
<td>1a Function coverage</td>
<td>+</td>
<td>+</td>
<td>++</td>
<td>++</td>
<td>Part of the code coverage report</td>
</tr>
<tr>
<td></td>
<td>coverage metrics at the</td>
<td>1b Call coverage</td>
<td>+</td>
<td>+</td>
<td>++</td>
<td>++</td>
<td>Part of the code coverage report</td>
</tr>
</tbody>
</table>
Qualification of software tools in the context of ISO26262

TargetLink and EmbeddedTester Qualified for ISO26262
ISO 26262: Software Tool Qualification

- **Tool Confidence Level (TCL):** defines need for qualification and appropriate measures
- **Tool Impact (TI):** impact of tool errors on the software/system
- **Tool Error Detection (TD):** probability of preventing or detecting tool errors

**Tool Classification**

- **Tool Impact**
  - TI 0
  - TI 1

- **Tool Error Detection**
  - TD 1
  - TD 2
  - TD 3
  - TD 4

- **Tool Confidence Level**
  - TCL 1
  - TCL 2
  - TCL 3
  - TCL 4

**Tool Qualification**

- Qualification for TCL 4
- Qualification for TCL 3
- Qualification for TCL 2
- Qualification not required
Code Generator Qualification for ISO 26262

TargetLink

- TCL based on Reference Workflow
- “Fit-for-Purpose” Certification

Tool Classification

- Tool Impact
- Tool Error Detection
- Tool Confidence Level

Tool Qualification

- Qualification for TCL 4
- Qualification for TCL 3
- Qualification for TCL 2
- Qualification not required

Analysis of tool use case
Test Tool Qualification for ISO 26262

BTC EmbeddedTester

- TCL based on Reference Workflow
- Certification: Validation of Software Tool and Evaluation of Tool Development Process

Tool Classification

- Tool Impact
  - TI 0
  - TI 1
- Tool Error Detection
  - TD 1
  - TD 2
  - TD 3
  - TD 4
- Tool Confidence Level
  - TCL1
  - TCL2
  - TCL3
  - TCL4

Qualification for TCL 4
Qualification for TCL 3
Qualification for TCL 2
Qualification not required

Analysis of tool use case

Model-Based Software Development for Safety-Critical Systems
TargetLife Reference Workflow
**ISO/DIS 26262 (highly) recommends**
- Back-to-Back test between Model and Code
- Structural Coverage Metrics for Software-Unit-Testing

**ISO/DIS 26262 demands Tool-Qualification**
- Also for Testing Tools used for revealing errors

**Embedded Tester offers**
- Automated Back-to-Back tests between MIL/SIL/PIL
- Different Structural Coverage Metrics up to MC/DC

**Qualify Embedded Tester for the automated Application of Back-to-Back Tests and Structural Coverage Measurement in 26262-compliant Processes**
Qualification with Validation Suite

- In 26262 Method „Validation of the Software Tool“ on ASIL-D is considered as „highly recommended“
- ”Validation of the software tool can be automated largely by using a validation suite.“ [ISO/DIS 26262-8]

- A Validation Suite (VS) contains
  - Feature Specifications for the relevant Features
  - Test Specifications for these Features incl. Feature Coverage
  - Test Implementation for the Test Specifications

- Qualification is achieved by executing the VS at the User‘s site
  - Added value: this approach also assures Quality of the Tool to be qualified in the User‘s environment
Conclusions

Benefits of an ISO 26262 Conform Model Based Development and Testing Process
Benefits of Model Based Development

- Development of *embedded systems* is a *time and cost* consuming procedure under growing *time-to-market* and new *quality and safety* standards pressure.
- **Model-Based Development** and **Autocoding** of safety-relevant software is widely applied for gaining *efficiency*.
- ISO 26262 explicitly **acknowledges** the paradigm of Model-based development with Autocoding to improve quality and ensure the safety needs.
- **TÜV approved** that **TargetLink** and **EmbeddedTester** are **fit** for purpose to develop and test safety-related software according to **ISO 26262, IEC 61508 and derivative standards**.

TargetLink

EmbeddedTester

ISO 26262
Benefits of advanced and integrated test method

- Functional testing finds about 20-40% of the problems.

- 30-40% of the software problems can be directly found by using the structural testing and back-to-back comparison.

- Formal verification is relevant for testing of safety-relevant software as it finds additional problems that might not be found by traditional testing methods.
Thanks for your attention!

- dSPACE and BTC Embedded Systems through DynaFusion in India can be your trustful partners in providing ISO 26262 conform products and know-how.

- We are looking forward to contacting you!