ABSTRACT

Model-based software development is increasingly being used to develop software for electronic control units (ECUs). When developing safety-related software, compared to non-safety-related software development, additional requirements specified by relevant safety-standards have to be met. Meeting these requirements should also be considered to be best practices for non-safety-related software.

This paper introduces a model-based reference workflow for the development of safety-related software conforming to relevant safety-standards such as IEC 61508 and ISO 26262. The reference workflow discusses requirements traceability aspects, software architecture considerations that help to support modular development and ease the verification of model parts and the code generated from those model parts, and the selection and enforcement of modeling and coding guidelines. Special focus is put on software unit and integration testing as an integral part of the overall verification and validation task. The presented methods and measures are mapped to the requirements of relevant functional safety-standards applied in the automotive industry.

Furthermore the paper discusses the role of such a reference workflow for answering the ongoing question for software tool qualification. The new automotive safety-standard ISO 26262 introduces a new approach to addressing this topic. This approach is based on a new way of classification of the software tool based on the tools impact and the degree of confidence that a malfunction or erroneous output can be prevented or detected.

SAFETY STANDARDS

Standards that apply to automotive software development are IEC 61508 [1] and particularly new ISO 26262 [2]. IEC 61508 is a generic across-the-industries standard that encourages the derivation of industry-specific standards. It originated in the process control automation industry, and sector-specific standards have already been derived for the process industry (IEC 61511), nuclear power plants (IEC 61513) and machinery (IEC 61513). New ISO 26262, which reached ISO Draft International Standard (DIS) status in July 2009, is a derivative that is especially tailored to the automotive industry. Where IEC 61508 has to remain abstract in many points, ISO 26262 is far more specific with regard to automotive electronics development. IEC 61508 was created and published in the late 1990s, before model-based development and code generation became widely adopted. It can therefore give little direct advice of how to comply within a model-based development process. The standard has therefore to be interpreted.

ISO 26262 AND MODEL-BASED DEVELOPMENT

ISO 26262 addresses this issue and specifically covers model-based development aspects, reflecting the importance of this approach in automotive software development today. The ISO 26262 part relevant to model-based development is “Part 6: Product development: software level”. It contains a separate chapter in the annex that describes the concept of model-based development of in-vehicle software and outlines its implications on the product development at the software level. In this annex also differences between code-based and model-based development are pointed out. Furthermore there are several notes throughout ISO/DIS 26262-6 directly
referring to specifics of model-based development. Examples are notes with regard to software unit design and implementation (ISO/DIS 26262-6, 8.5.1)

• NOTE In the case of model-based development, the implementation model specifies the software units in conjunction with other techniques (see Table 8).

or notes with regard to software unit testing (ISO/DIS 26262-6, 9.4.4)

• 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.

• 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.

MODEL-BASED REFERENCE WORKFLOW

When developing safety-related software, compared to non-safety-related software development as well as compared to code-based development, additional requirements and specifics described in the relevant safety-standards have to be met. In this situation a reference workflow can provide guidance for meeting the safety requirements of ISO 26262 and IEC 61508 to develop software up to and including ASIL D and/or SIL 3.

Based on best practices and experiences from real-world projects as well as taking the safety-requirements from the standards and the above mentioned ISO 26262 notes on model-based development into account a reference workflow for model-based development of safety-related software has been prepared for the established tool chain MATLAB®/Simulink®/Stateflow® and TargetLink [3]. This reference workflow describes model-based development including automatic code generation and model-based testing methods. Figure 1 shows the general elements of processes following this reference workflow. The outline addresses design- and implementation together with appropriate test and verification aspects:

• Textual requirements are designed and implemented in an executable model, which then itself is translated to code using code generation. Both steps are accompanied with appropriate guidelines.

• The step from textual requirements to a model ready for code generation is verified by performing model simulation and requirement-based testing, while the generated code is verified against the model through back-to-back testing, directly comparing the functional behavior of model and code. The test execution on model and code comes along with structural coverage measurement to assess the completeness of the tests and to avoid unintended functionality.

The key element of this workflow is the verification of the automatic conversion of the model into ECU program code. In order to demonstrate that the automatically generated code correctly implements the model the generated code shall be tested against the model by means of back-to-back testing. Many of the proposed methods are directly recommended by the standards themselves. It contains detailed reference tables that show how those methods and the overall workflow map to IEC 61508 and ISO 26262. The reference workflow has been approved by TÜV, an independent German certification
Users applying model-based development methods can directly relate to the reference workflow and demonstrate how the different aspects and methods are followed in the safety-related development project. Deviations from the methods described in this reference document are allowed as long as they are justified and documented.

REQUIREMENTS TRACEABILITY

Before software development itself can start, ISO 26262 asks for planning of the activities, methods and measures applied in the single subphases of software development, always considering the ASIL of the system under development. One important aspect to consider already upfront is traceability of requirements.

Requirements traceability refers to the ability to describe and follow the life of a requirement, in both forwards and backwards direction [3]. The goal is to follow a requirement to its implementation and its tests. Requirements traceability is helpful in order to show that requirements are fulfilled and have been tested. Traceability of requirements also helps ensuring completeness of requirements: by identifying requirements that are not considered in the model and by identifying model parts that cannot be linked to a requirement. The latter helps preventing the modeling and implementation of unintended behavior. Furthermore it facilitates the management of changes of requirements.

Requirements based development and verification is asked for by a number of software and safety-standards.

A major part of requirements traceability lies in the modeling environment providing the bidirectional, navigable links from external requirements management tools to the model. In order to achieve full traceability the code generator must establish the links between the model as input and the code as output. TargetLink as a code generator provides links between model and code that also support tracing requirements, for example by generating C code in HTML format that includes hyperlinks to the model.

MODELING AND CODING GUIDELINES

Another aspect to consider before software development itself can start is the selection of modelling and coding guidelines. ISO 26262-6 recommends the use of design and coding guidelines for modeling as well as programming languages. In general, but particularly for safety-related applications guidelines describing good programming style and avoiding unsafe language features should be used.

Modeling guidelines play an important role in ensuring good design quality as models progress from the initial function design to implementation model. Moreover they can help to achieve quality objectives with regard to functional safety of
the generated code. For the example tool chain MATLAB, Simulink, Stateflow and TargetLink mentioned before several, well established guideline documents exist:

- MathWorks Automotive Advisory Board guidelines (MAAB) [2], a collection of rules following several objectives such as increasing readability, smoothening workflows, and enabling design for verification and validation as well as for code generation
- TargetLink modeling guidelines [6] that cover the whole range from function development to production code generation
- MISRA TargetLink guidelines for the application of TargetLink in the context of automatic code generation, short MISRA AC TL [7].

At the beginning of a project the guidelines to be followed have to be defined. Guidelines from the above standard guideline documents should be selected. This selection can be enhanced by project-specific guidelines, for example to cover special naming conventions. It shall be documented which guidelines are to be followed. Committing to guidelines is only the first step. The second is to ensure - and document - that they are followed. Rule-based guideline checkers help to keep to the guidelines formally defined. Guideline Checkers are used to ensure and document that the models used in the project comply with the formally defined modeling guidelines. They can be applied early-on in the projects and allow to also checking bigger models efficiently.

On code level ISO 26262-6 specifically mentions MISRA C [8] as a suitable standard for C. At the same time ISO 26262 acknowledges that guidelines for automatically generated code and manual code can be different and MISRA itself specifically permits deviations from the standard as long as they are well justified. Therefore a comprehensive compliance document to the MISRA C guidelines for TargetLink is available. This document describes in detail whether a rule is always met, whether a rule is met only if certain conditions are observed on model level, whether the code generator can be configured in order to comply with a rule, or whether a rule is only partially fulfilled if the rule contains multiple code requirements.

In order to demonstrate compliance, similar to the model level also on the code level, standard commercial off the shelf MISRA C compliance checker tools can be used. Those tools not only check the resulting generated C code itself, but are also applied to any legacy or handwritten code that is part of the model. When static checking is performed on the generated code, detected violations have to be compared to the known and accepted violations of the code generator.

SOFTWARE ARCHITECTURE DESIGN
The software architecture design needs to consider design principles such as modularity and encapsulation, low complexity and maintainability and must be suitable for the subsequent software unit design and implementation. Modeling environments such as Simulink support the hierarchical and modular partitioning of models. Tools that allow the measurement of model complexity on system and subsystem level provide further help to achieve suitable module sizes. Code generators such as TargetLink allow modularization on code level by giving the user various means to configure whether model parts should be realized as separate functions, separate C-code files, etc.. Incremental code generation allows even smaller parts of the model to be coded separately with the benefit that changes in other parts of the model do not affect incrementally generated code that has already been verified.

The considerations described in the reference workflow are intended to support modular development and ease the verification of model parts and the code generated from those model parts. The basics of model and code verification remain unchanged applying these software architectural considerations.

MODEL-BASED TESTING
The test process accompanying the model-based development process - also referred to as model-based testing - benefits from the existence of an executable model and the simulation capabilities of the modeling environment. Systematic use of those simulation capabilities enables developers to perform fast, simple checks on the results obtained and on the modifications and adjustments that have been made during the development process.

Viewing the reference workflow from a test and verification perspective, the first significant activity is the verification of the model by demonstrating that the model is correct, meets its requirements and does not contain unintended functionality. Model verification is mainly done via simulation by performing functional, requirements-based tests. Test cases that cover all functional requirements have to be derived and executed. Another option would be to apply formal verification techniques. The second significant step following model verification is the verification of the generated code by demonstrating that the behavior of the code running on the ECU correctly implements the behavior of the verified model and does not contain any unintended functionality. This step assures that the automatic conversion of models into program code using automatic code generation preserves the behavior and does not introduce any errors. A valid method to demonstrate this is performing back-to-back tests between the simulation model and the generated code.

Back-to-back testing means testing the model and then re-testing the software using the same test cases and scenarios on model and code level, and comparing the results. All tests derived to verify the model, the test cases specified to cover
the functional requirements as well as the test cases to demonstrate structural coverage, shall be applied for back-to-back testing. The test stimuli are first applied to the model (MIL simulation). The results obtained serve as the reference. Then the same test stimuli are used to execute the object code derived from the generated code. The results of this execution are compared to the reference results obtained during model simulation. Execution of the resulting object code shall take place in an environment that matches as far as possible the ECU on which the code will be deployed. The actual target compiler used in the project shall be used for the translation of the generated code. The resulting object code shall be executed on a platform, e.g. an evaluation board, which contains the processor used in the ECU. PIL simulation offers such an environment.

The expressiveness of requirements-based and back-to-back testing depends on the completeness of the test cases. To evaluate the completeness of the test cases full coverage of the requirements is necessary and structural coverage of the model and code need to be measured. Portions of a model or code not covered by tests help revealing weaknesses of the completeness of the tests and detecting unintended functionality in the model or code.

**SOFTWARE TOOL QUALIFICATION**

All relevant international functional safety standards including IEC 61508 and ISO 26262 ask for evidence of software tool suitability and recommend the use of qualified tools.

In contrast to IEC 61508 or RTC DO-178B where tools are categorized by their nature and independent of their use in a concrete project ISO 26262 introduces a new way of classification based on the analysis of the software tool’s use-case in the concrete project. First the impact of the tool is determined by evaluating if a malfunctioning software tool and its erroneous output can lead to a violation of a safety requirement and thus lead to a failure of the system that affects its functional safety. Secondly the degree of confidence that such a malfunction or erroneous output can be prevented or detected in the project is analyzed. Based on the impact of the software tool and the tool error detection probability the so called tool confidence level is determined. And based on the tool confidence level the necessary tool qualification activities are derived considering the criticality (ASIL) of the system.

Figure 3. Principle of back-to-back tests within model-based development
TOOL CONFIDENCE LEVEL FOR CODE GENERATION TOOLS
The tool impact of a code generator is T11 meaning that an error might indeed cause the violation of a safety requirement. This requires the determination of the tool error detection which depends on the development workflow (“tool use case”) that is being used. At this point the model-based reference workflow introduced in this paper is very helpful. According to the TUV following the model-based reference workflow and its proposed verification and validation activities provides a high degree of confidence that a malfunction or erroneous output of the code generator can be prevented or detected. In this case the resulting tool confidence level is TCL1 and tool qualification for the code generator can be claimed without further tool qualification measures.

TOOL CONFIDENCE LEVEL FOR TEST TOOLS
ISO/DIS 26262 explains that the tool confidence level of a test tool can be the same or even higher than the one of a code generation tool. This becomes clear when taking a look at the reference workflow. Since the reference workflow heavily relies on model-based testing for the verification of the model as well as the verification of the generated ECU program code the use of an appropriate testing tool supporting these steps is crucial. In case the test tool doesn’t work correctly and produces erroneous output, a possible code generation error might not be detected. Therefore a high degree of confidence in the correct behaviour of the test tool needs to be established. Depending on the required tool confidence level and the ASIL of the system under test ISO 26262 suggests different methods for the qualification of a software tool.

EmbeddedTester [9] is an example for a testing tool that has been qualified for use in safety-related applications through validation of the software tool, a method suggested by ISO/DIS 26262 and suitable up to and including ASIL D. It supports model-based testing including back-to-back tests between model and code as well as measuring coverage to evaluate the completeness of the applied tests for the tool chain referred to in this paper.

SUMMARY
Model-based development is a suitable approach for the development of safety-related systems that has been successfully applied in several projects in the industry [10]. The orientation towards a standard-compliant and TUV approved reference workflow supports argumentation in the project that the chosen development approach and applied verification and validation measures fulfill the requirements of the safety standards to ensure a sufficient and an acceptable level of safety. The use of certified or qualified software tools provides further benefit to the user because additional, labourious tool qualification measures can be omitted.

REFERENCES
7. MISRA AC TL: Modeling style guidelines for the application of TargetLink in the context of automatic code generation, 2007, Version 1.0
8. MISRA-C: 2004 Guidelines for the use of the C Language in critical systems, MIRA, 2004
10. TargetLink - Driving the Future with Autocode, dSPACE Magazine Special Edition, 2009

CONTACT INFORMATION
Michael Beine
dSPACE GmbH
mailto:mbeine@dspace.de