A Model-Based Reference Workflow for the Development of Safety-Critical Software

A. Michael Beine[1]

1: dSPACE GmbH, Rathenaustraße 26, 33102 Paderborn

Abstract: Model-based software development is increasingly being used to develop software for electronic control units (ECUs). The automatic conversion of models into program code for ECUs plays a major role in this because it ensures efficient implementation, providing considerable savings potential and short development cycles.

This paper introduces a model-based reference workflow for the development of safety-critical software conforming to relevant safety-standards such as IEC 61508 and ISO 26262. The reference workflow provides guidance for meeting the safety requirements to develop software up to and including SIL 3 and/or ASIL D.

Furthermore the paper shows how such a reference workflow can help address the issue of software tool qualification.

Keywords: Model-based development, Reference workflow, Safety-critical systems, Safety standard, ISO 26262, IEC 61508, DO-178B

1. Introduction

Model-based development and automatic code generation have become increasingly established in recent years. The automotive and aerospace industry have widely adopted and successfully deployed these methods in many different series production programs worldwide. This brought various benefits, such as a reduction in development times, improved quality due to more precise specifications, and early verification and validation by means of simulation.

Model-based development is a general-purpose development approach which can be applied to a wide variety of applications. Safety-critical systems, which are largely found in aerospace applications, but also increasingly in the automotive industry, impose special additional requirements on this process. This leads to the question of how model-based design and automatic production code generation can be applied to the development of safety-critical systems. In order to answer this question the relevant safety standards need to be consulted.

2. Safety Standards

2.1 Overview

Standards that apply to automotive software development are IEC 61508 [1] and particularly the 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 were already derived for the process industry (IEC 61511), nuclear power plants (IEC 61513) and machinery (IEC 61513). The 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.

The US standard for developing airborne software is RTCA DO-178B; the European equivalent is ED-12B. IEC 61508 and DO-178B were published and revised during the 1990s before model-based design and automatic production code generation became common development approaches. They can therefore give little direct guidance on compliance within a model-based development process. The standards have therefore to be interpreted.

Aerospace industry and government experts are working on DO-178C, which will specifically address model-based development. The automotive industry now has ISO 26262.

2.2 ISO 26262 and model-based development

ISO 26262 addresses 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 for product development at the software level. The annex also details differences between code-based and model-based development. There are also several notes throughout ISO/DIS 26262-6 directly referring to specifics of model-based development. For example, there are notes on 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)*.

and notes on 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 behavior of the models with regard to the test objectives is equivalent to the automatically generated code*.

3. Model-Based Reference Workflow

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 when developing safety-critical software. In this situation a reference workflow can provide guidance for meeting the safety requirements of ISO 26262, IEC 61508 or RTCA DO-178B in developing software up to and including ASIL D, SIL 3 or Level A respectively.

Based on best practices and experience from real-world projects, and taking into account the safety requirements from IEC 61508 and ISO 26262 – including the ISO 26262 notes on model-based development mentioned above – a reference workflow for the model-based development of safety-critical 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.

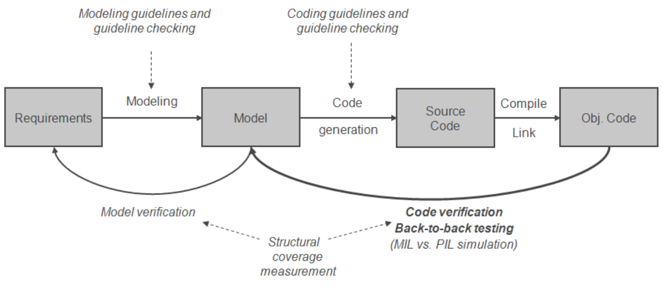
Modeling guidelines and guideline checking
Coding guidelines and guideline checking
Requirements
Modeling
Model
Code generation
Source Code
Compile Link
Obj. Code
Model verification
Code verification Back-to-back testing (MIL vs. PIL simulation)
Structural coverage measurement

Figure 1: Elements of the model-based reference workflow. Back-to-back testing between model and code is the key element in code verification.

Figure 1 shows the general elements of processes following this reference workflow. The outline addresses design and implementation, as well as appropriate testing and verification.

Textual requirements are designed and implemented in an executable model, which then is itself translated into code using code generation. Both steps are covered by 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 by back-to-back testing, directly comparing the functional behavior of the model and code. The test execution of the model and the code includes 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 must be tested against the model by means of back-to-back testing.

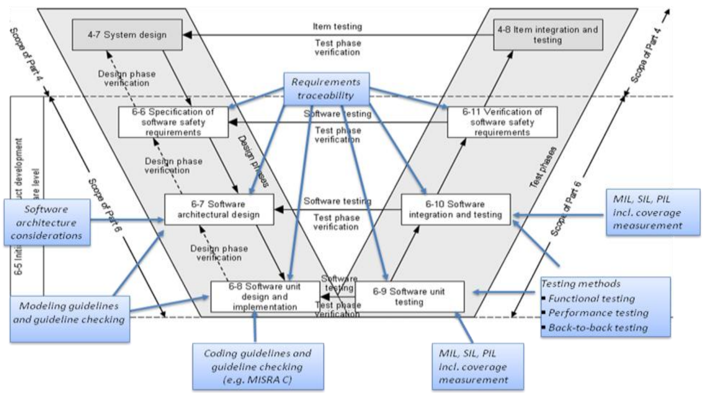
4-7 System design
Item testing
Test phase verification
4-8 Item integration and testing
Design phase verification
Requirements traceability
6-6 Specification of software safety requirements
Software testing
6-11 Verification of software safety requirements
Software architecture considerations
6-7 Software architectural design
6-10 Software integration and testing
MIL, SIL, PIL incl. coverage measurement
6-8 Software unit design and implementation
6-9 Software unit testing
Modeling guidelines and guideline checking
Testing methods
Functional testing
Performance testing
Back-to-back testing
Coding guidelines and guideline checking (e.g. MISRA C)
MIL, SIL, PIL incl. coverage measurement
Scope of Part 4
Scope of Part 6
Design phases
Test phases

Figure 2: Elements of the reference workflow mapped to the ISO 26262-6 reference phase model for software development.

Many of the proposed methods are directly recommended by the standards themselves. Figure 2 shows a rough mapping. Additionally, the reference workflow contains detailed reference tables that show how the methods and the overall workflow map to IEC 61508 and ISO 26262. The reference workflow has been approved by TÜV, an independent German certification authority. 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-critical development project. Deviations from the methods described in this reference document are allowed as long as they are justified and documented.

3.1 Requirements Traceability

Before software development itself can start, ISO 26262 requires the planning of the activities, methods and measures used in the individual subphases of software development, always with reference to 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 directions [4]. The goal is to track a requirement to its implementation and its tests. Requirements traceability is helpful in determining whether requirements have been fulfilled and tested. The traceability of requirements also helps ensure their completeness, by identifying requirements that are not included in the model and by identifying model parts that cannot be linked to a requirement. The latter helps prevent the modeling and implementation of unintended behavior. It also facilitates the management of requirements changes. Requirements-based development and verification are stipulated by a number of software and safety standards.

A major part of requirements traceability lies in the modeling environment, which provides 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 the model and code that also support the tracing of requirements, for example by generating C code in HTML format that includes hyperlinks to the model.

3.2 The Role of Guidelines

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

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

- The MathWorks Automotive Advisory Board guidelines (MAAB) [5], a collection of rules with 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, or MISRA AC TL for short [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. These can be supplemented by project-specific guidelines, for example, special naming conventions. A record must be made of which guidelines are to be followed. Committing to guidelines is only the first step. The second is to ensure – and document – that they are being followed. Rule-based guideline checkers help to keep to the formally defined guidelines. Guideline checkers are used to ensure and document that the models used in the project comply with the modeling guidelines. They can be applied early on in projects and also allow large models to be checked efficiently.

Coding guidelines – 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. A comprehensive compliance document for TargetLink is available as a supplement to the MISRA C guidelines. 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 it contains multiple code requirements.

To demonstrate compliance on the code level in a similar way to the model level, standard commercial off-the-shelf MISRA C compliance checker tools can be used. Those tools check not only the resulting generated C code itself, but also 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.

3.3 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 subsequent software unit design and implementation. Modeling environments such as Simulink support the hierarchical, modular partitioning of models. Tools for measuring model complexity on system and subsystem level provide further help in achieving 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 elements described in the reference workflow are intended to support modular development and facilitate the verification of model parts and the code generated from them. The basics of model and code verification are not changed by applying these software architecture considerations.

3.4 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 made during the development process.

Viewing the reference workflow from a testing and verification perspective, the first significant activity is to verify the model by demonstrating that it 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 to verify 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 ensures that converting models into program code by means of automatic code generation preserves its 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 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 and the test cases to demonstrate structural coverage must be used in 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.

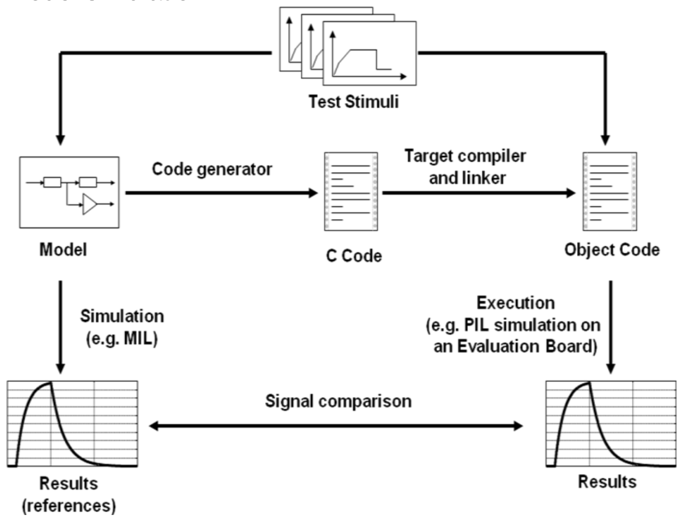

Figure 3: Principle of back-to-back tests in model-based development

The resulting object code should be executed in an environment that matches as closely as possible the ECU on which the code will be deployed. The actual target compiler used in the project should be used to translate the generated code. The resulting object code should be executed on a platform, e.g., an evaluation board, which contains the processor used in the ECU. Processor-in-the-loop (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 the structural coverage of the model and code needs to be measured. Portions of a model or code not covered by tests help reveal weaknesses in the completeness of the tests and detect unintended functionality in the model or code.

4. Software Tool Qualification

All relevant international functional safety standards, including ISO 26262 and DO-178B, require evidence of software tool suitability and recommend the use of qualified tools.

One of the first questions that are often asked when using model-based development and automatic code generation for safety-critical application development is the one about tool qualification: Is it possible to use a non-certified, non-qualified tool for the development of safety-critical software? The answer is: Yes, that is common practice in the industry today – for compilers, for linkers, and also for code generators. TargetLink, for example, has been successfully used for years in several safety-critical projects in the automotive as well as in the aerospace industry [10].

4.1 Tool Qualification and DO-178B

RTCA DO-178B clearly states that a qualified tool is required only if there is no complete verification of the tool's output. More precisely, according to DO-178B (12.2), tool qualification has to be performed only if process steps described in DO-178B are eliminated, reduced or automated.

DO-178B distinguishes between "development tools" and "verification tools". The major distinction criterion is a tool's direct impact on the software product. A development tool's output will be part of the airborne software, and therefore errors introduced by that tool introduce errors in the end product. A verification tool's output will not be part of the airborne software; however, an error in a verification tool can lead to non-detection of an error in the airborne software.

In a model-based development process the controller code is produced by a code generator which is clearly in the category of "development tools". Qualification of a development tool according to DO-178B is an option that is rarely exercised since it is not practical from a managerial point of view and not easy from a technical point of view as [12] explains.

The main advantages of using non-qualified development tools are that it is possible to use patch releases and new functionality without having to wait for the requalification of patch versions, as well as the fact that no additional tool costs (qualified tools can be significantly more expensive) and no extra tool qualification costs are associated with them [12]. Using a non-qualified code generator and applying a standard-compliant software verification process in order to verify the code generator output is an attractive alternative. In this case, a DO-178B-compliant, model-based reference workflow similar to the one described in this paper is an argument in favor of using a non-qualified code generator for safety-critical software development.

Verification and validation in a model-based development process to a large part rely on automated testing. Since complete verification of the test results is not practical, the use of a qualified verification tool is advisable. In contrast to the qualification of development tools, the qualification of a software verification tool according to DO-178B is almost common practice. Qualification criteria according to DO-178B (12.2.2) are met by "demonstrating that the tool complies with its Tool Operational Requirements under normal operational conditions".

To fulfill this criterion, test suites that cover a tool's operational requirements and can be executed by the user in the context of the development project are available from tool vendors. This approach is similar to but less demanding than the validation suite approach explained below for qualification of a test tool according to ISO 26262, where the reaction of the tool under anomalous operation conditions, for example, also needs to be examined.

4.2 Tool Qualification According to ISO 26262

In contrast to IEC 61508 or RTC DO-178B, where tools are categorized by their nature and independently of their use in a concrete project, ISO 26262 introduces a new method of tool classification based on an analysis of a software tool's use case in a 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 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. The so-called tool confidence level is determined from the impact of the software tool and the tool error detection probability. Then the necessary tool qualification activities are derived from the tool confidence level with reference to the criticality (ASIL) of the system.

If there is a high degree of confidence that a malfunction or an erroneous output from the software tool will be prevented or detected, i.e., if there is a high tool confidence level, no additional qualification measures are required. In all other cases additional measures are required to demonstrate that the software tool fulfills its use cases with the required level of confidence. ISO 26262 suggests and details four possible methods:

- Increased confidence from use
- Evaluation of the development process
- Validation of the software tool
- Development in compliance with a safety standard

<u>Qualification of a code generation tool</u> – The tool impact of a code generator is TI1, 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 TÜV, 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.

Qualification of a test tool – 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 is clearly shown by the typical model-based development workflow. Since the 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 to support these steps is crucial. If the test tool does not work correctly and produces erroneous output, a possible code generation error might not be detected. Since there is typically no systematic verification of the results in subsequent development phases, tool error detection level TD4 has to be assumed. This results in a low level of confidence as defined by ISO 26262, thus requiring additional measures to establish a high degree of confidence in the correct behavior of the test tool.

EmbeddedTester [9] is an example of a testing tool that has been qualified for use in safety-critical applications through a combination of *validation of the software tool* and *evaluation of the development process,* and thus is 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 tests used in the tool chain referred to in this paper.

5. Conclusion

Following a standard-compliant and officially approved reference workflow not only provides guidance and orientation for the user, it also helps demonstrate that the chosen development approach and applied verification and validation measures fulfill the requirements of the safety standards to ensure a sufficient and acceptable level of safety.
For DO-178B projects, demonstrating that a software verification process is standard-compliant allows the use of non-qualified development tools such as a code generator.

With regard to the automotive industry, the reference workflow supports tool qualification according to ISO 26262. The reference workflow serves as a detailed description of a development workflow as required by ISO 26262. for the analysis of the software tool use case in the project. In fact the TargetLink reference workflow used as an example in this paper was an important factor in the certification of TargetLink for IEC 61508 and ISO/DIS 26262 applications by TÜV SÜD, a German certification authority.

6. References

[1] Functional Safety of Electrical / Electronic / Programmable Electronic Safety Related Systems, IEC 61508, 1998

[2] Road vehicles – Functional Safety, International Organization for Standardization, ISO 26262 (Draft International Standard), 2009

[3] Model-Based Software Development for Safety-Related Systems, TargetLink Reference Workflow, Version 1.1, Michael Beine, 2010

[4] Gotel, O., and A. Finkelstein: *“An Analysis of the Requirements Traceability Problem”*, Proceedings of the First International Conference on Requirements Engineering, Colorado Springs, Colo., April 1994, pp. 94-101.

[5] The MathWorks Automotive Advisory Board, Control Algorithm Modeling, Guidelines using MATLAB®, Simulink®, and Stateflow®, Version 2.0, 2007

[6] Modeling Guidelines for MATLAB / Simulink / Stateflow and TargetLink Version 2.1, dSPACE GmbH, 2008

[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

[9] EmbeddedTester, http://www.btc-es.de/

[10] TargetLink - Driving the Future with Autocode, dSPACE Magazine Special Edition, 2009

[11] Beine, Otterbach, Jungmann: *“Development of Safety-Critical Software Using Automatic Code Generation”*, SAE World Congress, 2004-01-0708, 2004

[12] Kornecki, Zalewski: *“The Qualification of Software Development Tools From the DO-178B Certification Perspective”*, Cross Talk – The Journal of Defense Software Engineering, April 2006

7. Glossary

MIL:	Model-in-the-Loop
PIL:	Processor-in-the-Loop
ASIL:	Automotive Safety Integrity Level
SIL:	Safety Integrity Level
ECU:	Electronic Control Unit
TI:	Tool impact
TD:	Tool error detection
TCL:	Tool confidence level