10 Steps to ISO26262-compliant Model-based Software Development

Hartmut Lackner and Heiko Dörr
Model Engineering Solutions GmbH
Mauerstr. 79, 10117 Berlin, Germany
{hartmut.lackner | doerr}@model-engineers.com

Abstract

Testing of large software systems is a major challenge for the automotive industry. All test activities, particularly for safety-critical components, must be performed in adherence to the ISO26262 standard. The development team of a software project faces a multitude of requirements on the testing process to achieve ISO26262-compliance. These requirements are defined in a generic way such that an interpretation before implementation is required – and of course, they shall be implemented in cost-minimal ways.

We propose a 10-step strategy to achieve an ISO26262-compliant testing process. This strategy relates ISO26262 requirements with state-of-the art methods and approaches for virtual testing of software components. It is based on a prioritization of the requirements of ISO26262 with regard to their impact on the quality of the software. We conclude the contribution with considerations for automation.

1 Introduction

Model-based software development is a standard software development approach for automotive embedded software [3]. Enhancing a software development process to comply with ISO26262 is a challenge. One challenge represents the safety standard itself: as a standard, it defines abstract, general requirements and leaves it to the software development team to assess their respective relevance to the project and implement them. This assessment is necessary as ISO26262 considers both code-based and model-based software development and contains requirements that are applicable for only one of both approaches. In practice, interpretation, assessment and implementation of requirements presuppose comprehensive knowledge of state-of-the art methods, tools available on the market, and individual project conditions.

How to make the best use of the often very tight project budget is made difficult by uncertainty regarding the advantages and disadvantages of methods and corresponding tools. Further issues that need to be addressed concern the clarification and assignment of roles and responsibilities as well as the definition of interfaces between different teams and departments.

For most software development teams, there is no reasonable chance of addressing the outlined questions all at once, due to insufficient project resources. Hence, we provide a suitable strategy for prioritizing the requirements based on an assessment of the necessary actions. We weigh up the impact of an activity onto the quality of the developed software and the costs for introducing and applying it. Hence, the main contributions of this paper are two-fold: 1) an interpretation of the ISO26262 requirements for testing; 2) a prioritization of the ISO26262 requirements that allows ISO26262-compliance to be efficiently achieved.

The remainder is structured as follows: In Sect. 2, we briefly introduce the foundation of model-based design as applied in the automotive industry. In Sect. 3, we present the requirements and our partitioning into three groups. Then, we present our 10-step strategy in Sect. 4, before we conclude in the final section.

2 Foundations

In the automotive domain, the approach for developing embedded software has changed in recent years: today, model-based development provides an efficient approach for the software development of embedded systems, where executable graphical models are used at all development stages. Figure 1 illustrates the model-based development process in a simplified way. Here, functional software models are used to verify functional requirements as an executable specification, and also as so-called implementation models used for target code generation. The models are designed with common graphical modeling languages, such as Simulink and Stateflow from The MathWorks [2] which are used in combination with code generators such as TargetLink by dSPACE [1] or the Real-Time Workshop/Embedded Coder by The MathWorks. A comprehensive survey of quality assurance for model-based development is given in [4].

Most frequently used means for safe-guarding model-based development are: guideline checking and testing models. Modeling guidelines, e.g. enforce use of modeling elements compatible to the employed code generator or that error-prone modeling elements are avoided. In testing models, model verification (model-in-the-loop, MiL) ensures that the model behavior fulfills the requirements; and code verification (software-in-the-loop, SiL) shows equivalence of code and model,

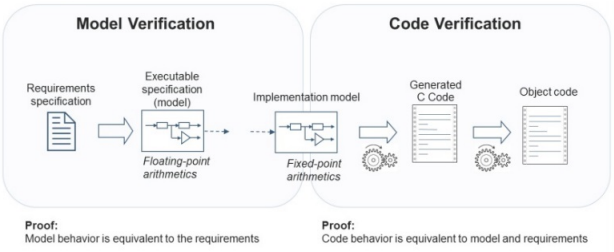

Figure 1: Model-based Development Process

which also implies fulfillment of requirements.

3 ISO26262 Requirements on Software Development

A key element within part 6 of ISO26262 is a V-model based software development process with well-known development phases as shown in Figure 2. For each of the development phases, a set of requirements is defined. Practically speaking, the requirements in part 6 of ISO26262 bring nothing technically new to software development but rather are requirements on ensuring quality of software.

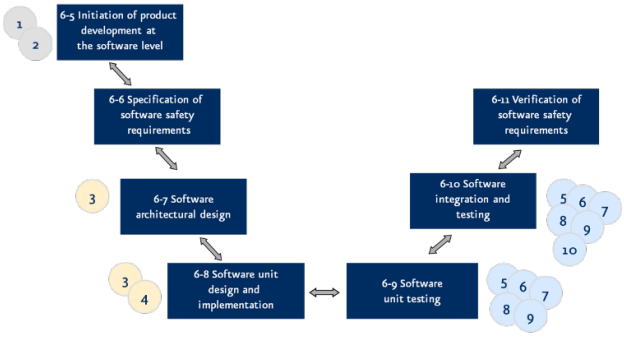

Figure 2: Model-based development process with annotated steps in circles.

We group those requirements into measures to implement a *systematic development process*, measures for *constructive quality assurance* and measures for *analytical quality assurance*. Measures for constructive quality assurance are used to build quality in. The main goal of constructive quality assurance is the avoidance of failures. A typical example for constructive quality assurance is design guidelines that are employed to avoid error-prone design patterns. Measures for analytical quality are used to identify quality issues such as detection of implementation failures. Testing is part of analytical quality assurance.

We summarized the ISO26262 requirements for our strategy into ten steps. The name of each step points out the focus of the activities necessary to implement the requirements. Also, we present the steps in groups of the measures as identified above.

Systematic Development Process

1. Tailor development process [6.5]
2. Adjust tool chain [6.5]

Constructive Quality Assurance

3. Create design specifications [6.7, 6.8]
4. Check and review models [6.8]

Analytic Quality Assurance

5. Test focus: requirements [6.9, 6.10]
6. Back-to-back tests [6.9, 6.10]
7. Test focus: interfaces [6.9, 6.10]
8. Test focus: robustness and resources [6.9, 6.10]
9. Test focus: error guessing [6.9, 6.10]
10. Check unspecified functionality [6.10]

4 10-Step Strategy

The strategy presented here, reflects our experiences supporting our customers in achieving ISO26262 compliant model-based software development processes. One of our key observations is that defining an efficient development process is a step-by-step process of introducing new methods and tools, working with these tools in real projects, and defining rules and guidelines based on experiences gained. This is why we propose a bottom-up approach for defining a suitable development process, starting with *detecting failures*, then *avoiding failures*, and finally implementing a *compliant process*. In the following, we present the steps in prioritized order, including our own experiences and references to the required activities by the standard.

4.1 Failure Detection

We consider ISO26262 requirements aimed at detecting failures in the model as the most important. These requirements are contained in step 1 Test focus: requirements, step 2 Back-to-back tests and step 3 Test focus: interfaces. According to our experience, we can expect the most significant benefit with regard to the quality of the model to come from testing. For this reason, we propose to fulfill the ISO26262 for requirements-based testing, interface testing and back-to-back testing first.

1 Test focus: requirements Requirements-based testing allows the systematic detection of implementation failures and of failures due to incomplete or inconsistent requirements (6-9.4.3 T10-1e, 6-10.4.3 T13-1e). An indispensable precondition for requirements-based testing is documented requirements and a defined requirements management. The coverage between requirements and test specifications and the structural coverage are necessary to validate the test specification (6-9.4.5, T12-1a/1b). Only the combination of test results and coverage data allows reliable statements regarding the presence or absence of failures in the model

2 Back-to-back tests This step ensures that the code generated from the model complies with the model (6-9.4.3 T10-1e, 6-10.4.3 T13-1e). This is achieved by re-executing the tests in either SiL or PiL mode and comparing these results with those from the MiL testing. The combination of MiL and back-to-back testing makes it easier to detect failures that depend on code-generation-specific model properties such as target-processor-specific data types. Furthermore, we propose measuring the requirements coverage in addition to the unit testing level, also on the integration testing level (6-10.4.5).

3 Test focus: interfaces ISO26262 requires systematic testing of interfaces to ensure they are implemented correctly (6-9.4.3 T10-1b, T11, 6-10.4.3 T13-1b, T14). We observe that addressing external and internal interfaces during testing explicitly allows the early detection of failures that would otherwise remain unnoticed until later integration testing phases, where it might be more difficult to localize them. For software with an ASIL C or D rating, ISO26262 requests application of coverage metrics for addressing the execution of the software functions (6-10.4.5, 6-10.4.6, T15). These coverage metrics must be assessed for integration tests but not for unit testing.

4.2 Failure Avoidance

The second group (step 4 to 6) includes mainly ISO26262 requirements that contribute to avoiding failures. The benefit is different compared to the first group of requirements. These constructive quality assurance methods contribute to avoiding specific classes of failures. Not only do they have a positive impact on the quality of the model and the generated code, but they also improve the efficiency of the development process. Modeling guidelines contribute to avoiding subtle failures such as unnoticed erroneous data type conversions or missing initializations. Such failures may require a lot of time for analyzing after their detection in the test phase. Modeling guidelines improve the efficiency by additionally improving the readability and maintainability of models. A consistent model layout considerably reduces the time for review and failure analysis.

4 Check and review model Required activities are the application of modeling guidelines (6-5.4.7-T1, 6-8.4.4-T8) and manual reviews (6-8.4.5-T9-1a/1b). We have noticed that these activities are crucial as according to our experiences, models are used, modified and enhanced over quite long periods of time. Our frequently used approach for defining a guideline set consists of starting with an initial guideline set and applying it to the available models. Based on the results, the guideline set is then adapted e.g. by removing guidelines, replacing guidelines with alternative guidelines or parameterizing guidelines according to project needs. This procedure contributes to the discussion concerning the relevance of guidelines because it takes into account company-specific modeling practices right from the beginning.

Concerning the manual reviews, we propose the execution of a formal review independent on the ASIL level of the software unit as according to our experience this will achieve better results, provided that it is based on a review checklist.

5 Test focus: error guessing and static code analysis For error-guessing, the test engineer makes assumptions about possible failures of the software and defines tests to address them (6-9.4.4, T11, 6-10.4.4,T14). A skilled tester uses experiences from previous projects and modeling-language specific knowledge to define the test cases, hence this method is also known as experienced-based testing.

Also measuring MC/DC coverage for ASIL D rated components is grouped into this step, since ISO26262 does not prescribe a fixed percentage for MC/DC but a *sufficient percentage* (6-9.4.5, T12). This requirement may seem too imprecise, but defining a fixed value would make the handling of variants in the software almost impossible. Of course, a rational must be provided for the actual MC/DC coverage value, if it deviates from 100%.

Furthermore, static code analysis is required for such components (6-8.4.5, T9-1g). To employ static code analysis in practice, it should be restricted with regard to types of failures and in particular parts of the software. Otherwise, static analysis can give unclear results if there are not sufficient resources for signing off irrelevant warnings.

6 Create design specifications For ISO ISO26262 compliance, design specifications are needed for both the software architecture (6-7.4.1-7.4.5, T3) as well as for software units (6-8.4.2-8.4.4, T8). Based on our experience, there is more need to describe the software architecture due to the increased complexity of the functionality. We have noticed that the number of failures due to incorrect interactions of software units has grown. As they cannot be detected with unit tests, such failures are detected late and their correction can be difficult and expensive. The software architectural design contributes to avoiding these failures by providing an overview on the overall software and explaining details on the context of the software units

For model-based development, the model itself can be the unit specification. In this case, it should be combined with supporting documentation describing aspects that are not clearly defined by the model.

4.3 Implement Compliant Process

The last group of requirements contained in steps 7 to 10 mainly summarizes the previous steps and aims at combining them into an efficient development process supported by a seamless tool chain. The doc-

umentation of the development process and in particular the documentation of the interpretation of ISO26262 requirements are of particular importance. Documenting the decisions concerning the implementation of requirements relieves the development teams from time-consuming reinterpreting of the ISO26262 requirements and allows them to focus on the development and quality assurance activities.

7 Check unspecified functionality ISO26262 requires analyzing of the software with respect to unspecified functionality (6-10.4.7). Unspecified functionality includes for example instrumentation code or deactivated functions due to software variants. ISO26262 does not require deletion of this code, but it must have no impact on the safety requirements. This activity can be addressed with corresponding tests but also with manual reviews. Then, an argument must be provided to explain how the deactivation is ensured at runtime.

8 Adjust Tool Chain According to our experience, advanced development teams have strict guidelines and documentation for the usage of their tools, e.g. they use fixed code generator settings, since the search for failures resulting from different configurations is time-consuming. Furthermore, ISO26262 requires the qualification of tools: Such a qualification consists of an assessment of a tools capability to introduce failures and the potential for them to remain unnoticed. Depending on the outcome of this assessment, it may be necessary to e.g. validate the tool with comprehensive tests.

9 Test Focus: Robustness & Resources Although software is correct, a system may fail due to hardware faults or incorrect usage of the software (6-9.4.3, T10-1c,1d, 6-10.4.3, T13-1c,1d). Resource usage tests validate non-functional properties of the software such as execution times or resources such as memory. Hence, executing these tests on a target processor is a necessity. Fault injection tests address the robustness of the software, i.e. they aim to validate sufficient failure handling and can be executed in any test environment.

10 Tailor Development Process We propose to perform the tailoring of the development process similar to a lessons learned session. Therefor, the sequence, dependencies and safety-relevance of the development activities should be analyzed in order to define the development process. There is no restriction concerning how the content of the ISO work products is combined in or spread over documents. Although for ISO26262 compliance, a mapping of the work products onto those required by the standard is mandatory.

Our experience has shown that the documentation of the development process is of particular importance. This should be the company-specific interpretation of ISO26262, i.e. it should clearly define what exactly needs to be done for ISO26262 compliance. In other words, reading the process definition should be sufficient for knowing how to proceed in the development of software with an ASIL rating. We believe that investing time and effort in the definition and documentation of the development process is crucial. A defined process with company specific regulations concerning the implementation of ISO26262 requirements relieves the development team from considering the standard itself. The documentation of decisions regarding the interpretation of the standard allows the developers to concentrate on the development and quality assurance activities themselves.

5 Conclusions

In this paper we presented 10 steps how to achieve ISO26262 compliance for model-based software development projects. This strategy fulfills ISO26262 requirements by using state-of-the art methods and approaches which are today applied by major OEMs and suppliers of the automotive industry. The 10 steps strategy is based on a prioritization of the requirements of ISO26262 with regard to their impact on the quality of the software. We are convinced that such a prioritizing is necessary in order to cope with the budget constraints every team has to deal with. Our approach applies the project budget optimally for both: addressing possible gaps in terms of ISO26262 compliance as well as improving the quality of the software to be developed.

Furthermore, there is a wide range of tools readily available to facilitate the individual steps. Especially tools for testing, guideline checking, and static analysis offer many features for automating repetitive tasks, such as test execution, back-to-back testing, guideline checking, and standard-compliant reporting. In our experience, these tools perform more efficiently and are easier to use than internally developed solutions.

References

[1] dspace: Targetlink production code generator. `http://www.dspace.com/`. Accessed: 2017-01-06.

[2] The mathworks. `http://www.mathworks.com/products`. Accessed: 2017-01-06.

[3] ISO/TC 22. ISO/DIS 26262 road vehicles – functional safety, 2011.

[4] Ines Fey and Ingo Stürmer. Quality assurance methods for model-based development. In *SAE Technical Paper*. SAE International, 04 2007.