ANSYS

Technical Paper

A Cost-Effective Model-Based Approach for Developing ISO 26262 Compliant Automotive Safety Related Applications

Automotive manufacturers and their suppliers increasingly need to follow the objectives of ISO 26262 as it is now state-of-the art and as it is the case that an ever increasing number of active and passive safety systems are developed within cars. This has increased the need to define a safe system development process. This paper proposes a model-based approach including automatic and certified code generation to efficiently implement the embedded software that controls these systems while meeting the needed safety requirements and obeying the rules of ISO 26262.

This paper presents an approach for the development of ISO 26262 [1] compliant software applications based on Model-Based Design (MBD) and certified Automatic Code Generation (ACG). When using such a method, strong benefits can be achieved in reducing software verification costs while meeting the objectives of ISO 26262 at the highest ASIL levels.

After a brief summary of the ISO 26262 standard, the paper presents a detailed analysis of what is really expected by the standard for tool qualification, with strong emphasis on having a Safety Case that takes into accounts the various stakeholders, i.e. the tool developer, the tool installer and the tool user, considering all possible failure conditions and mitigation actions by performing hazard analysis and risk assessment using a method such as HAZOP [3].

According to the standard, a Safety Case has been built for the qualification of the Automatic Code Generator (ACG) and it has shown the various mitigation actions that should be taken. It is clearly demonstrated that if we want the tool users not to perform low level verification activities such as application software code reviews and code level testing, a number of precise actions have to be taken by the tool developer in order to provide the required tool confidence level.

This typically includes tool code reviews and tool structural code coverage. Therefore, any tool qualification method that would only rely on “validation of the software tool” cannot meet these objectives for the tool user. The methods that consist in performing “tool development in accordance with a safety standard” such as ISO 26262 or DO-178C [2] will allow us to meet the objectives that we have described above. However, ISO 26262 allows for an appropriate combination of methods to achieve this.

In a final section of the paper we present a complete SCADE MBD flow relying on the existence of the certified ACG and a few typical industrial use cases of safety-related applications such as advanced driver assistance systems (ADAS).

The Purpose, Organization and Requirements of ISO-26262

In a context where safety is one of the key issues in automobile development, the ISO 26262 standard provides for the development of safety-related systems:

- the definition of an automotive safety lifecycle
- a risk-based approach based on Automotive Safety Integrity Levels (ASIL) from D representing the highest level to A the lowest level
- and the use of ASILs for specifying the applicable requirements of ISO 26262 to be met while developing the system so as to avoid unreasonable residual risks.

Let us now briefly describe the following three topics in the standard:

- the ISO 26262 concept phase
- the ISO 26262 requirements for product development
- and the ISO 26262 tool qualification process.

The ISO 26262 Concept Phase

Part 3 of ISO 26262 describes the requirements of the concept phase of ISO-26262. Assuming an item has been identified, safety analyses are carried out on the basis of the item definition and the safety concept is derived from it.

The objective of the “Hazard analysis and risk assessment phase” is to identify and categorize the potential hazards of the item and formulate safety goals to prevent or mitigate hazards in order to achieve acceptable risk.

Situation analysis and hazard identification are performed in order to identify the potential unintended behaviors of the item that could lead to a hazardous event. Hazard classification involves the determination of the hazard severity (S) depending on potential harm, probability of exposure (E) depending on driving situation, and controllability (C) by the driver, as defined by Tables 1, 2 and 3 below.

Table 1. Classes of severity.

Class	S0	S1	S2	S3
Description	No injuries	Light and moderate injuries	Severe and life-threatening injuries (survival probable)	Life-threatening injuries (survival uncertain), fatal injuries

Table 2. Classes of probability of exposure regarding operational situations.

Class	E1	E2	E3	E4
Description	Very low probability	Low probability	Medium probability	High probability

Table 3. Classes of controllability.

Class	C0	C1	C2	C3
Description	Controllable in general	Simply controllable	Normally controllable	Difficult to control or uncontrollable

As a function of the item S, E, C, the ASIL-Level can be determined for each hazardous event according to the following Table 4. QM means Quality Management, i.e. no specific requirement according to ISO-26262.

Table 4. ASIL determination

		C1	C2	C3
S1	E1	QM	QM	QM
	E2	QM	QM	QM
	E3	QM	QM	A
	E4	QM	A	B
S2	E1	QM	QM	QM
	E2	QM	QM	A
	E3	QM	A	B
	E4	A	B	C
S3	E1	QM	QM	A
	E2	QM	A	B
	E3	A	B	C
	E4	B	C	D

Let us take an airbag example and consider two combinations of driving situation and hazardous events in Table 5 below.

Table 5. ASIL Determination for an airbag example.

Situation	Hazard	Severity	Probability	Controllability	ASIL
High-speed driving and crash	Airbag does not deploy	S3	E1	C3	A
High speed driving	Airbag deploys	S3	E4	C3	D

In this example we see the influence of the probability of exposure:

- In the first case, the probability to be in situation of having a crash while driving fast is very low, therefore the item that controls that the airbag deploys when it should can be assigned ASIL A (low).
- In the second case, the probability to be in situation of driving fast is high, therefore the item that controls that the airbag does not deploy when it should not must assigned ASIL D (high).

The ISO 26262 Requirements for Product Development

Part 6 of ISO 26262 describes the requirements for product development at the software level including requirements, architectural design, unit design and implementation, unit and integration testing.

Figure 1 below represents the system and embedded software V-cycle of ISO 26262. The activities related to system development are described in ISO 26262-4 (Part 4) and the ones related to embedded software are described in ISO 26262-6 (Part 6).

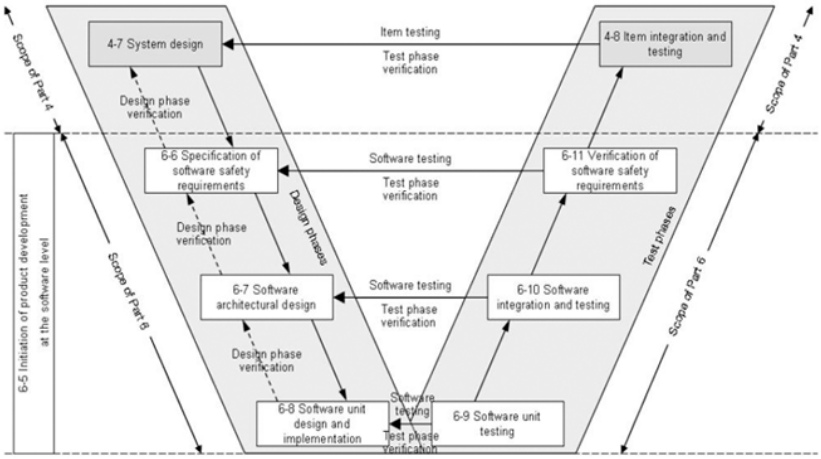


Figure 1. The system and embedded software lifecycle V-cycle in ISO 26262.

For each phase of the V-cycle, a number of tables have been created and contain methods listed to contribute to the required level if confidence in achieving compliance with the corresponding requirement.

The ISO 26262 Tool Qualification Process

Part 8 of ISO-26262 describes the requirements for supporting processes including management of safety requirements, configuration and change management, documentation, confidence in the use software tools, etc. In the context of this paper we will concentrate on the question of "confidence in the use of software tools" as tool qualification will be key to reducing the cost of developing and verifying the safety-related software.

Clause 11 of ISO-26262 provides criteria to determine the required level of confidence in a software tool and means of qualification of a software tool so that the tool user can rely on the correct functioning of the tool as it is used for achieving activities required by ISO 26262.

The intended usage of the software tool shall be analyzed to determine the Tool Impact (TI):

- TI1 shall be selected when there is an argument that there no possibility that a malfunction of the software tool can introduce or fail to detect and error in the safety-related software being developed.
- TI2 shall be selected otherwise.

The confidence in measures that prevent the software tool from malfunctioning or that detect that the tool has malfunctioned is expressed by the Tool error Detection (TD) class:

- TD1 shall be selected if there is a high degree of confidence that a malfunction can be prevented or detected.
- TD2 shall be selected if degree is medium.
- TD3 shall be selected otherwise.

Based on TI and TD, the tool confidence level (TCL) will be determined according to Table 6 below.

		Tool error detection		
		TD1	**TD2**	**TD3**
Tool impact	**TI1**	TCL1	TCL1	TCL1
	TI2	TCL1	TCL2	TCL3

Table 6. Determination of the tool confidence level (TCL).

For example, for a code generator producing source code from a software model and in the case the output of the code generator (the source code) is not verified, classes TI2 (tool may introduce an error) and TD3 (malfunction may not be detected) must be selected and therefore the tool confidence level is TCL3.

If we consider this example, ISO 26262-8 Clause 11 defines in Table 7 below the possible methods to qualify the tool as a function of the ASIL of the safety-related item that is being developed.

Table 7. Qualification of software tools classified TCL3.

Methods		ASIL			
		A	B	C	D
1 a	Increased confidence from use in accordance with 11.4.7	++	++	+	+
1 b	Evaluation of the tool development process I accordance with11.4.8	++	++	+	+
1 c	Validation of the software tool in accordance with 11.4.9	+	+	++	++
1 d	Development in accordance with a safety standard [a]	+	+	++	++
[a] No safety standard is fully applicable to the development of software tools. Instead, a relevant subset of requirements of the safety standard can be selected.					
EXAMPLE: Development of the software tool in accordance with ISO 26262, IEC 61508 or RCTA DO-178					

Therefore, if we assume the embedded software is ASIL D, there are two highly recommended (++) methods for qualifying the code generation tool:

- Validation of the code generator in accordance with 11.4.9 which demonstrates that the tool complies with its requirements, typically by running a test suite that evaluates the functional and non-functional aspects of the tool

- Development of the code generator in accordance with a safety standard (e. g., ISO 26262, IEC 61508, DO-178C).

Independently of the method that is used to qualify the tool, Section 11.4.10 of ISO 26262-8 requires that the tool is evaluated in accordance with Table 1 of ISO-26262-2 to ensure the correct evaluation of the required TCL and the appropriate qualification of the software tool in accordance with its required TCL. Table 1 requires that a safety assessment of the tool is performed according to clause 4 of ISO 26262-3.

Clause 4 of ISO 26262-3 requires that hazard analysis and risk assessment are performed in order to identify the potential hazards of the item (the tool) and formulate safety goals related to the prevention or mitigation of these hazards. This is precisely what we will do in the next Section of this paper.

Qualification of the SCADE Suite Code Generator

As illustrated in Figures 2 and 3 below, the SCADE Suite KCG code generator [5] takes as input a SCADE model made of a combination of state machines and data flows and produces as output an equivalent C program.

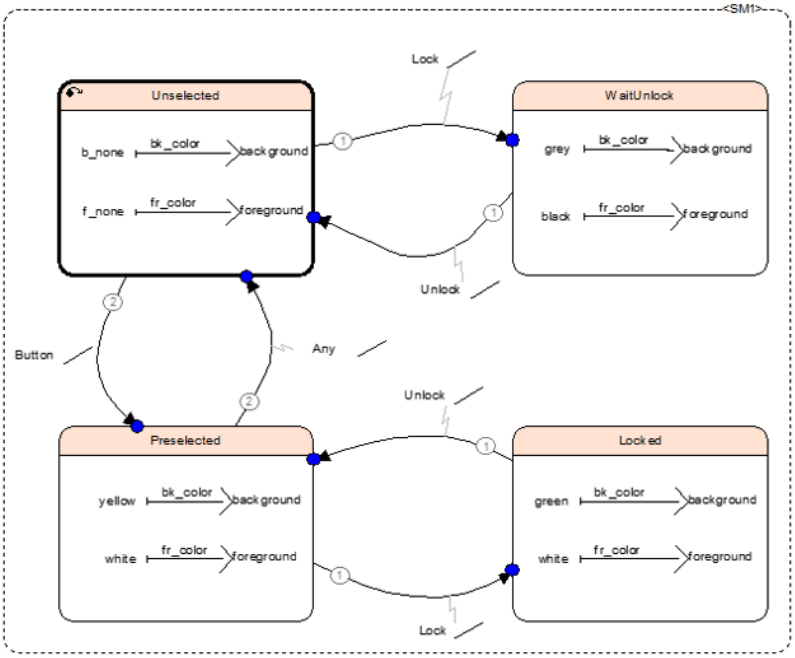
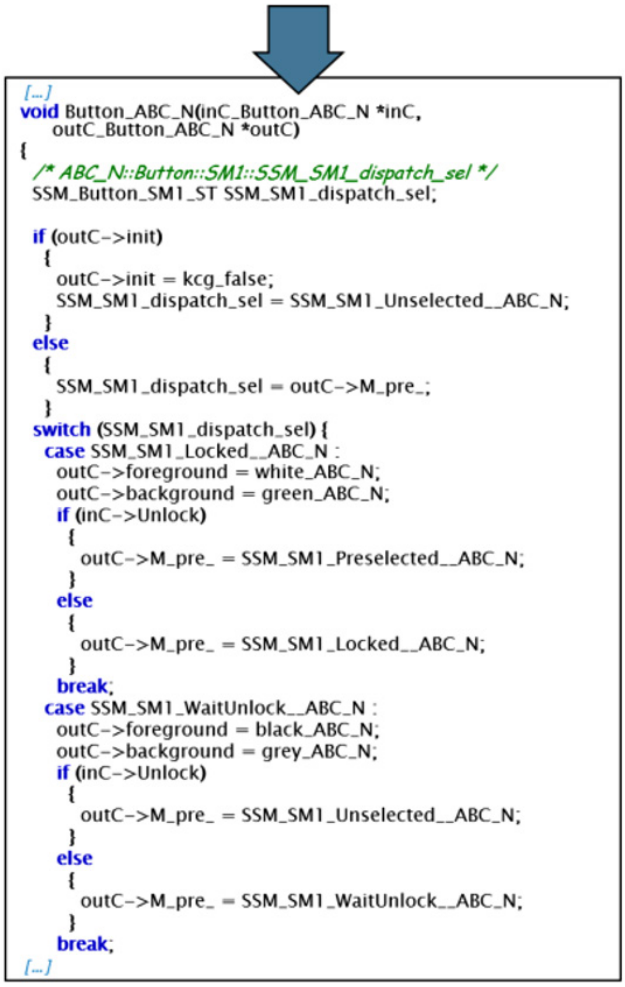


Figure 2. A SCADE model example.

The SCADE Suite KCG code generator has been qualified at TCL 3 by TÜV SÜD to be used to develop ASIL D automotive software. In order to achieve qualification, KCG has been developed following the IEC 61508 standard (i.e., method 1d above). As initially described in [6] in the context of the UK military standard Def Stan 00-56 [4], hazard analysis and risk assessment have been performed on the KCG tool using HAZOP, a method recommended by ISO 26262. This analysis is based on KCG architecture and usage process. It produces deviations, potential causes and mitigation actions from all tool stakeholders:

- Tool developer
- Tool installer
- and Tool user



```
[...]
void Button_ABC_N(inC_Button_ABC_N *inC,
    outC_Button_ABC_N *outC)
{
  /* ABC_N::Button::SM1::SSM_SM1_dispatch_sel */
  SSM_Button_SM1_ST SSM_SM1_dispatch_sel;

  if (outC->init)
   {
     outC->init = kcg_false;
     SSM_SM1_dispatch_sel = SSM_SM1_Unselected__ABC_N;
   }
  else
   {
     SSM_SM1_dispatch_sel = outC->M_pre_;
   }
  switch (SSM_SM1_dispatch_sel) {
   case SSM_SM1_Locked__ABC_N :
     outC->foreground = white_ABC_N;
     outC->background = green_ABC_N;
     if (inC->Unlock)
      {
        outC->M_pre_ = SSM_SM1_Preselected__ABC_N;
      }
     else
      {
        outC->M_pre_ = SSM_SM1_Locked__ABC_N;
      }
     break;
   case SSM_SM1_WaitUnlock__ABC_N :
     outC->foreground = black_ABC_N;
     outC->background = grey_ABC_N;
     if (inC->Unlock)
      {
        outC->M_pre_ = SSM_SM1_Unselected__ABC_N;
      }
     else
      {
        outC->M_pre_ = SSM_SM1_WaitUnlock__ABC_N;
      }
     break;
[...]
```

Figure 3. The generated code from SCADE Suite KCG for this example SCADE model.

On the basis of this safety argument, dozens of failure conditions have been identified and dozens of individual mitigation actions have been allocated to tool developer, tool installer and tool user. This is illustrated by examples in the Table 8 below.

User actions	Installer actions	Developer actions
Verification of user-supplied code, type specification, command line and customization	Verification of correct tool installation and OS version	Verification of command line and file processing
Validation of correct tool invocation and outcome	Verification of command line and file processing capabilities	100% tool requirements based test coverage
Use of the tool on multiple development systems		Verification of the absence of tool unintended functions
Verification of post-generation code modifications		100% MC/DC tool structural coverage
Analysis of object code properties		Analysis of tool response to resource constraints
		Analysis of underlying OS and library functions

Table 8. Examples of mitigation areas, classified by tool stakeholder.

This table shows that developer's actions have to be set in order to mitigate risks due to errors in the code generator.

For example, there is a risk that unintended functionality is present in the code generator and that, under some circumstances that would not be detected by requirements-based test cases of the code generator, this would lead to a catastrophic failure of the embedded software.

This is precisely the role of a typical safety process such as ISO 26262 or DO-178C to detect such cases. This is achieved by verification activities performed during the development cycle of the code generator such as:

- design and code reviews
- MC/DC structural coverage analysis [7]

As a conclusion to this Section, we may say that:

- We have clearly established what is required by the ISO 26262 standard when a TCL3 tool such as a code generator used with minimal error detec tion measures has to be qualified.
- The safety assessment that is mandated establishes that, in order to achieve significant benefits in reducing the code level verification activi ties for the tool user, a number of actions have to be performed by the tool developer.
- These actions are typically the ones that are requested by the safety stan dards such as ISO 26262, IEC 61508 or DO-178C (design and code re views, requirements-based testing and structural code coverage).
- These actions cannot be performed in the case the tool qualification method consists in running a test suite that merely evaluates the function al and non-functional aspects of the code generator. However, ISO 26262 allows for an appropriate combination of methods to achieve this.

On the basis of the availability of a code generator that has been quali-fied by using the method that is described above, we will now describe a complete toolset that allows an efficient implementation of the V-cycle of Figure 1.

An Efficient Model-Based Development Flow with SCADE

The complete solution that we present in Figure 4 below is based on the SCADE toolset [5], with SCADE System for system and software architecture, SCADE Suite for software components design and qualified automatic code generation, SCADE LifeCycle for traceability of requirements and SCADE Test for verification and validation.

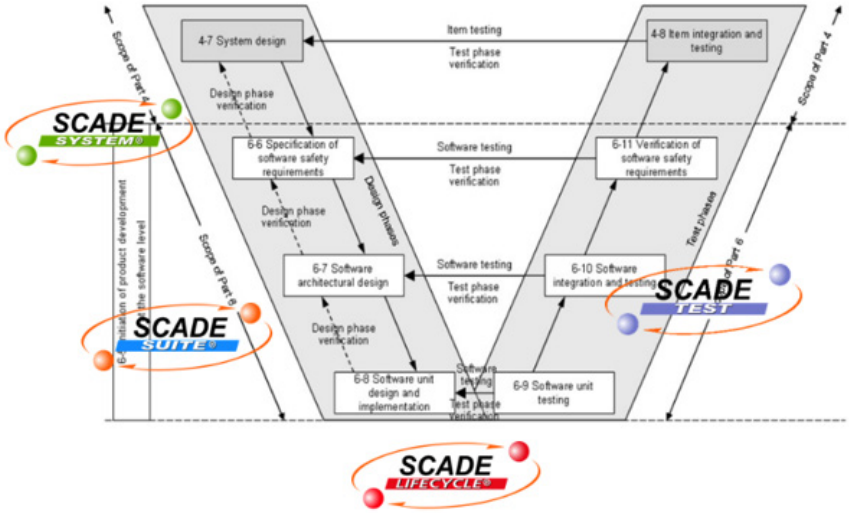


Figure 4: The System Design, Specification of Software Safety Requirements and Software Architectural Design phases of ISO 26262.

Let us now illustrate the various phases of this lifecycle.

System and Software Architectural Design

The SCADE System tool will be used as shown in Figure 5 below to describe the System and Software architecture.

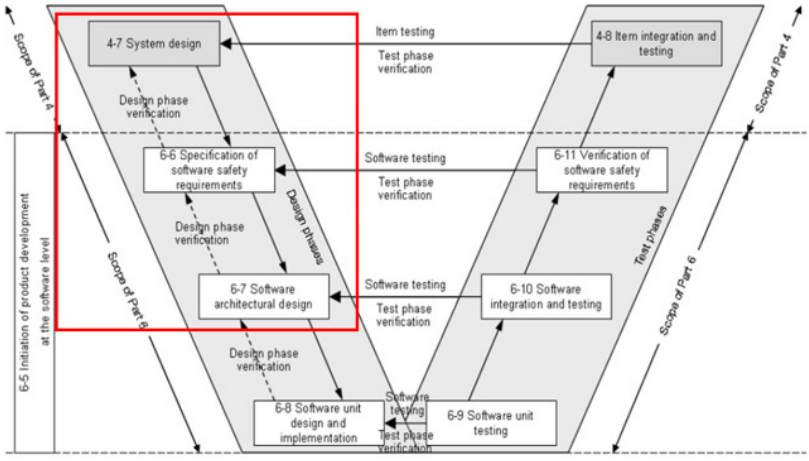
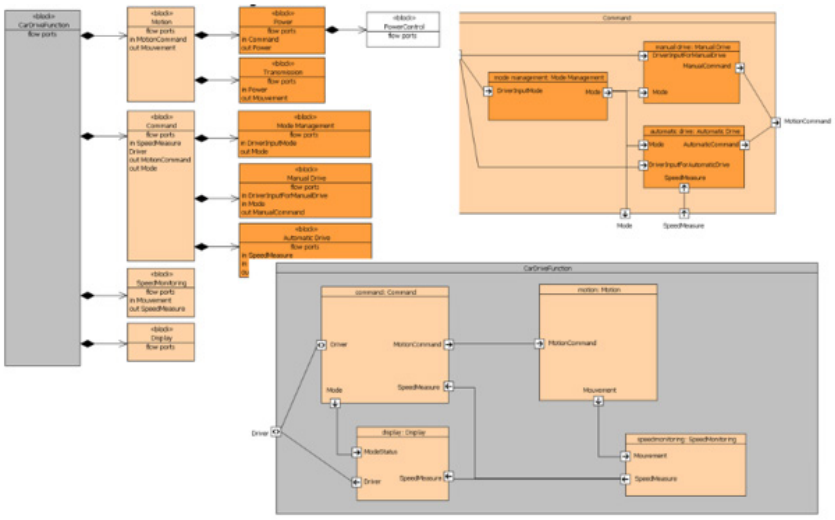


Figure 5. The System Design, Specification of Software Safety Requirements and Software Architectural Design phases of ISO 26262.

SCADE System is a SysML-based [8] tool that will typically be used to represent the System functional and architectural designs, as shown in Figure 6 below.



On top of describing the system functions and architecture, a traceability framework has to be set up at the start of the project. The SCADE LifeCycle product allows establishing all traceability links from the initial system requirements, to the design models, the generated code and the test scenarios.

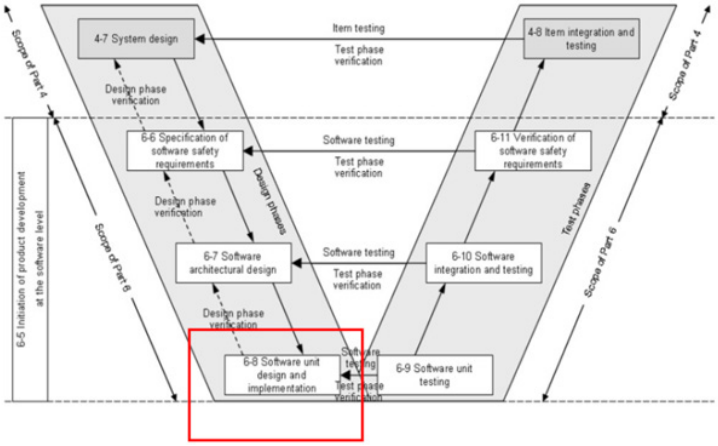


Figure 7: The Software unit design and implementation phase of ISO 26262.

Figure 8 below illustrates a typical SCADE Suite software design.

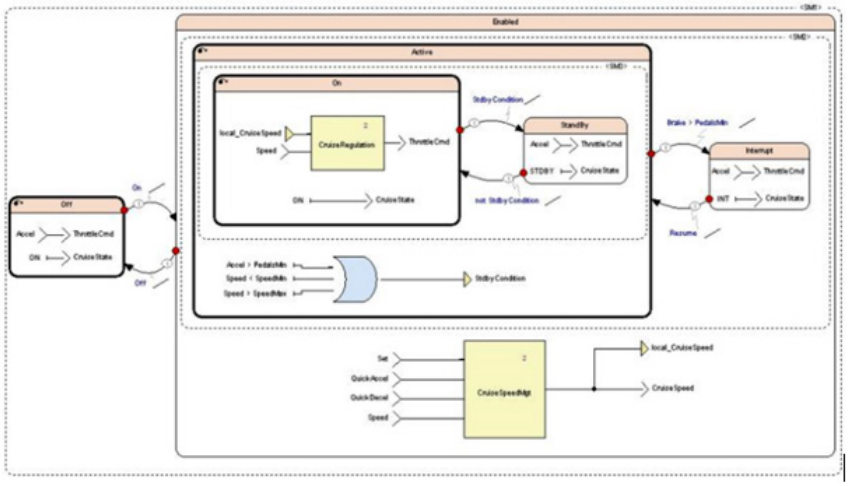


Figure 8: A SCADE Suite design description of a software module.

During this phase, initial verification activities regarding the software module will be performed. This includes:

- Semantics check of the SCADE Suite models
- Reviews of the SCADE Suite models that can be performed on the basis of the SCADE Suite reporter, a qualified tool
- Traceability analysis between the Software safety requirements and the SCADE Suite models through the use of the SCADE Application Lifecycle Management Gateway

Software Unit Testing and Integration Testing

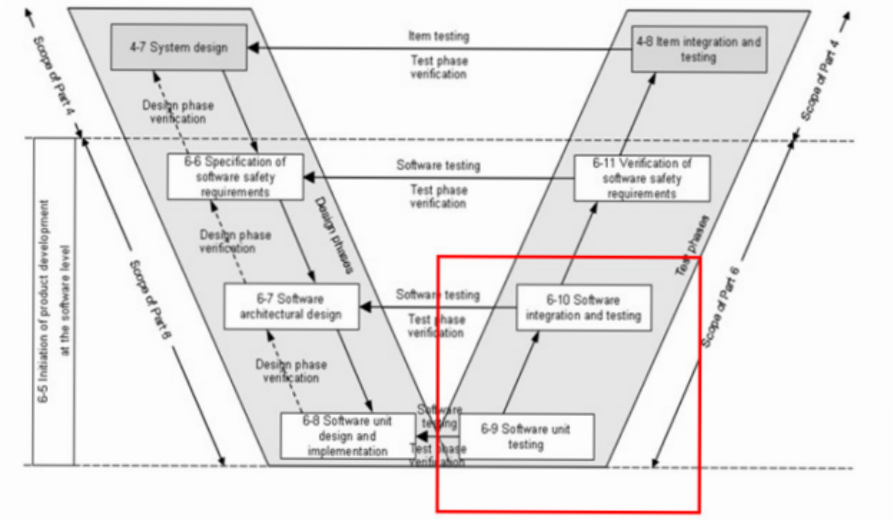


Figure 9: The Software unit testing and the Software integration and testing phases of ISO 26262.

Now moving to the software verification and validation activities, SCADE Test provides a number of model-level verification functionalities:

• Simulation of the SCADE Suite models through the use of the SCADE Test Environment for Host (as shown in Figure 10 below) in order to verify that the requirements-base tests that have been created produce expected results. This is recorded in a Host Conformity Report

• Automated rerun of the above test cases by the SCADE Test Environment for Target to verify that software execution on target still produces expected results. This is recorded in a Target Conformity Report.

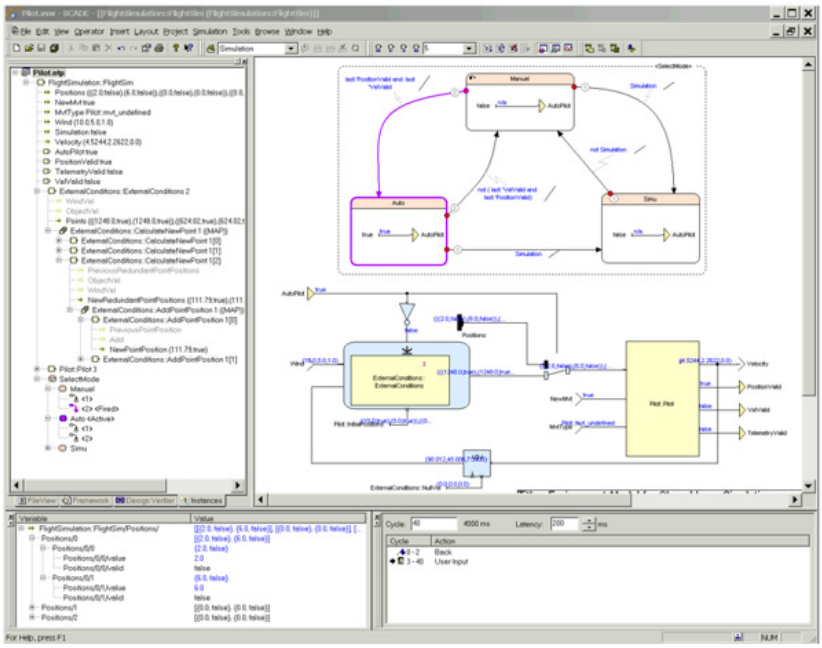


Figure 10: SCADE Suite simulation of a software module.

Finally, we have to demonstrate that the test cases that have been created from the safety requirements fully cover the SCADE model. The role of this activity is to verify the thoroughness of requirementsbased simulation/testing. Model coverage analysis can also contribute to the detection of unintended functionality. This is achieved by using SCADE Test Model Coverage, as illustrated by the Figure 11 below.

SCADE Test Model Coverage performs a MC/DC [7] assessment of the structural coverage of the SCADE Suite model by the requirements-based tests, together with an assessment of the generated code coverage, thus satisfying the objectives of ISO 26262 at the highest ASIL level.

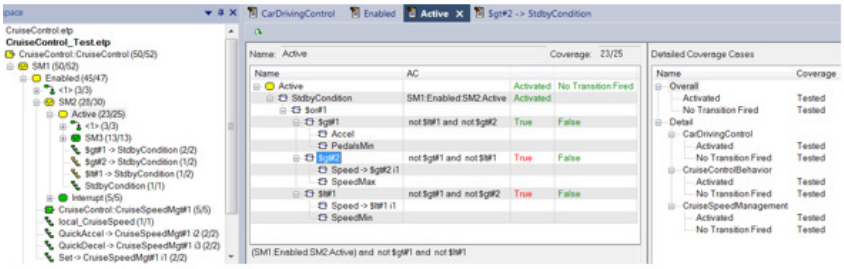


Figure 11: SCADE Test Model Coverage structural coverage assessment of a software module showing that 50 of the 52 required testing situations have been covered.

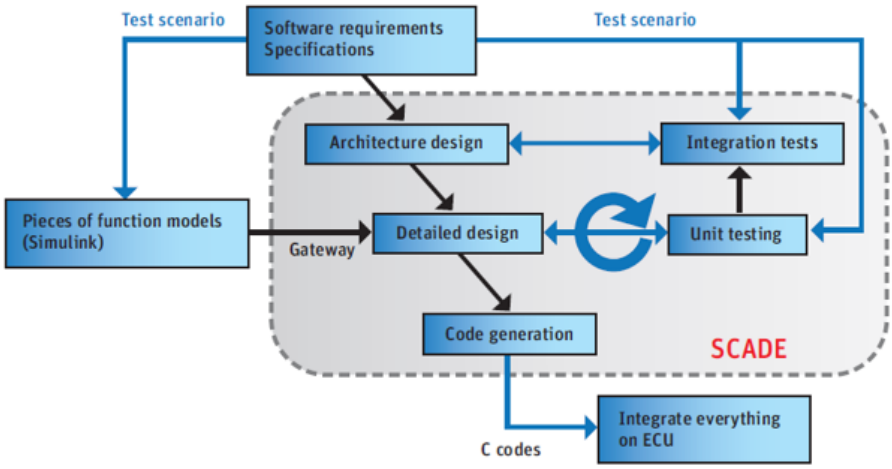


Figure 12: HEV software development process with SCADE at Suburu.

The Subaru Industrial Use Case

The methods and tools that are described in this paper have been used by Subaru to complete a large and very complex control application while significantly reducing software development and testing costs. This electric vehicle control application is fully described in [9].

The Figure 12 below describes the SCADE flow that is used by Subaru, including architecture design, detailed design, code generation and testing

Summary/Conclusions

This paper has first provided a detailed explanation of what is really expected by the ISO 26262 standard when qualifying a tool that can introduce errors into the embedded code. In particular it is shown that, independently of the qualification method that is chosen, a Safety Case regarding to the use of the tool has to be built so that it can be demonstrated that the tool qualification process is commensurate with the risks associated to the tool failures.

Such as safety case has been built for an automatic code generator (SCADE Suite KCG) and it shows quite clearly that a number of tool developer actions have to be done. This includes tool design and tool code reviews and structural coverage analysis of the tool code. This is precisely what is requested by the safety standards (ISO 26262 or DO-178C) that can be used to qualify a tool for ISO 26262 and this the method that we followed.

Finally, on top of the backbone of this approach, the qualified code generator, we have built a complete toolset including all needed verification tools. This complete toolset allows to efficiently meet all the objectives of ISO 26262 at the highest ASIL levels and it is shown how it has been used in an industrial use case.

Definitions/Abbreviations

ADAS - Advanced Driver Assistance System
ACG - Automatic Code Generation
ASIL - Automotive Safety Integrity Level
IEC - International Electrotechnical Commission
HAZOP - Hazard and operability study
ISO - International Organization for Standardization
MBD - Model-Based Development
MC/DC - Modified Condition/Decision Coverage
QM - Quality Management
SysML - Systems Modeling Language
TCL - Tool Confidence Level
TD - Tool error Detection
TI - Tool Impact

References

1. ISO, “ISO 26262 Road vehicles - Functional safety”, ISO, Nov. 2011.
2. RTCA, “DO-178C Software Considerations in Airborne Systems and Equipment Certification”, RCTA, Dec. 2011.
3. IEC, “IEC 61882:200 Hazard and operability studies (HAZOP studies) - Application guide”, IEC, 2001.
4. MOD, “Ministry of Defence, Defence Standard 00-56”, UK MOD, Jun. 2007
5. ANSYS, “SCADE R16 User Manual”, Jan. 2015.
6. Stephenson Zoë, Kelly Tim, Camus Jean-Louis, “Developing an Argument for Def Stan 00-56 from Existing Qualification Evidence”, Proceedings of ERTS 2010, May 19th-21st 2010, Toulouse. Downloaded from SAE International by Bernard Dion, Monday, March 28, 2016
7. Hayhurst Kelly, Veerhusen Dan, Chilensky John, Rierson Leanna, “A Practical Tutorial on Modified Condition/Decision Coverage”, NASA Technical Report TM-2001-210876, May 2001.
8. OMG, “Systems Modeling Language, Version 1.3”, OMG, Aug. 2011
9. Kurihara Masaru, “Safe Automobile Controls”, ANSYS Advantage, Volume VII, Issue 3, 2013

ANSYS, Inc.
Southpointe
2600 ANSYS Drive
Canonsburg, PA 15317
U.S.A.
724.746.3304
ansysinfo@ansys.com