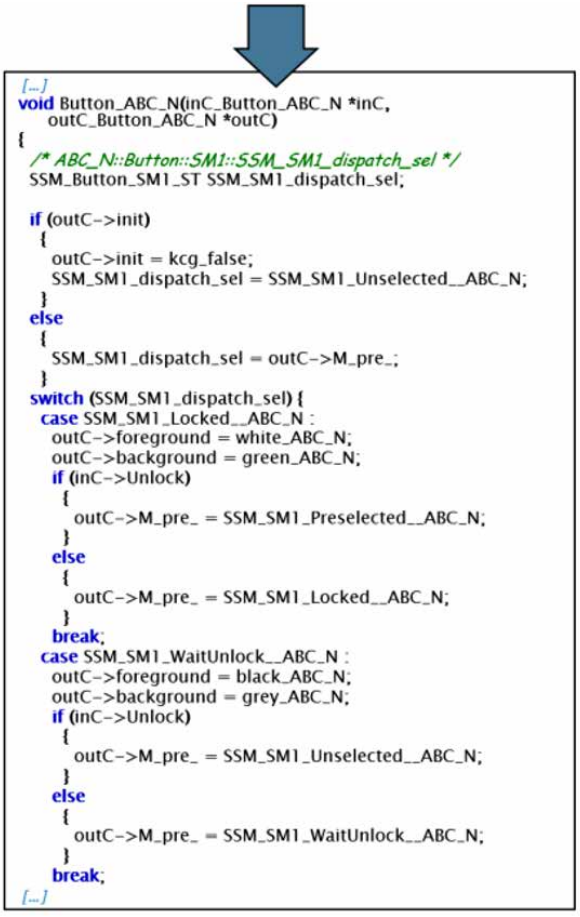
The SCADE Suite KCG code generator has been qualified at TCL 3 by TÜV SÜD to be used to develop ASIL D automotive software. In order to achieve qualification, KCG has been developed following the IEC 61508 standard (i.e., method 1d above). As initially described in [6] in the context of the UK military 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.

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 detection 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 activities 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 standards such as ISO 26262, IEC 61508 or DO-178C (design and code reviews, 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 functional 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 qualified 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.

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.

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.

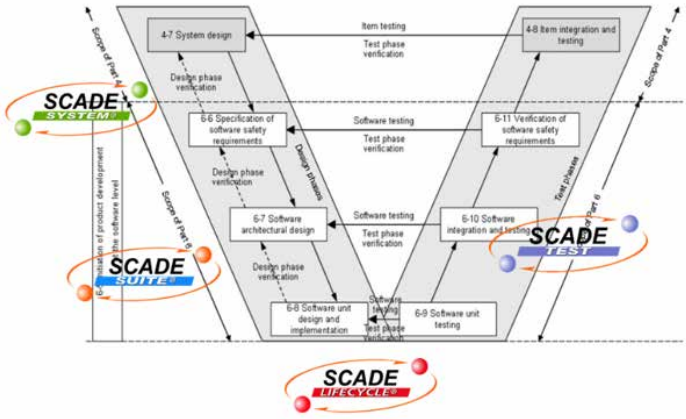


Figure 4: The System Design, Specification of Software Safety Requirements and Software Architectural Design phases of ISO 26262.

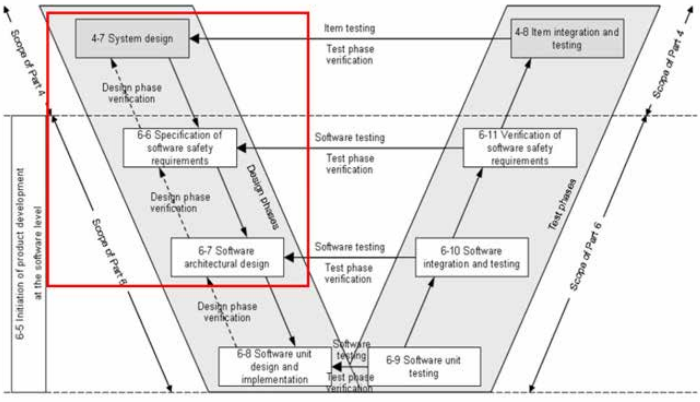


Figure 5. The System Design, Specification of Software Safety Requirements and Software Architectural Design phases of ISO 26262.

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.

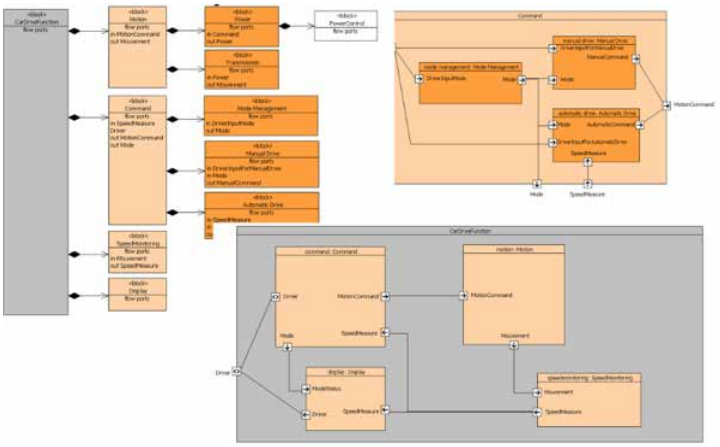


Figure 6

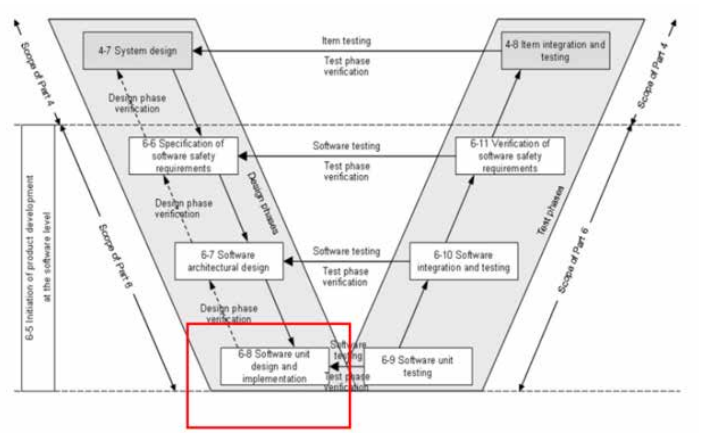


Figure 7: The Software unit design and implementation phase of ISO 26262

Figure 8 below illustrates a typical SCADE Suite software design.

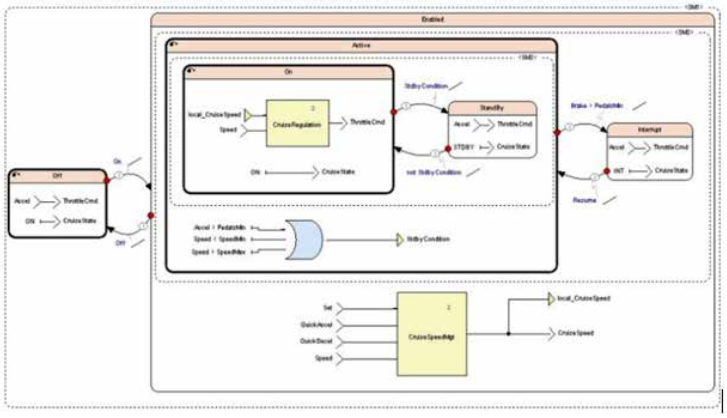


Figure 8: A SCADE Suite design description of a software module.

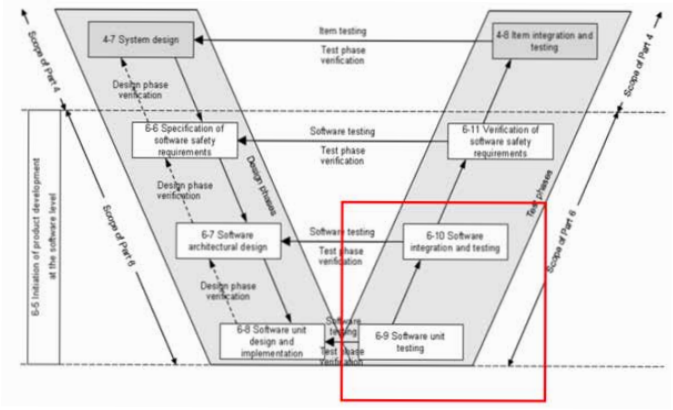


Figure 9: The Software unit testing and the Software integration and testing phases of ISO 26262.

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

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.

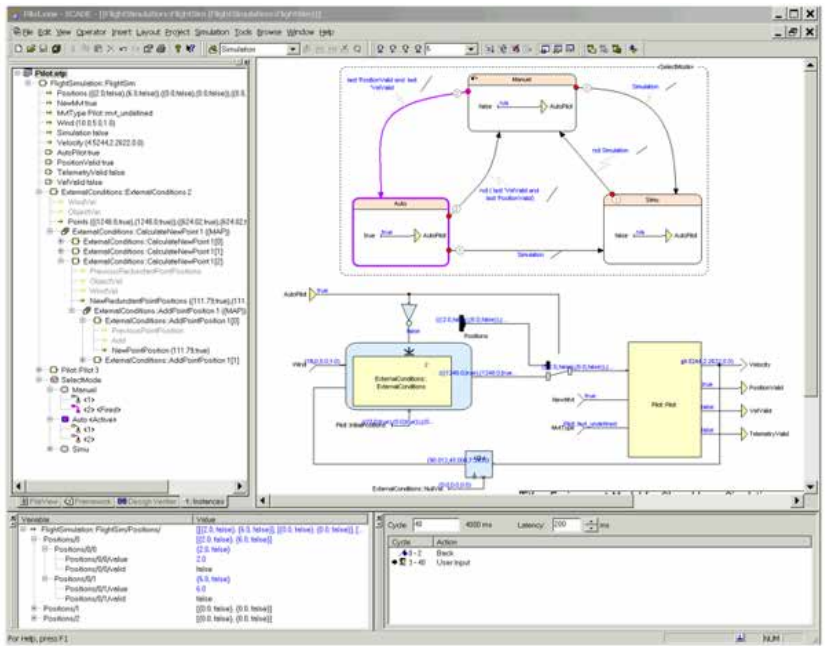


Figure 10: SCADE Suite simulation of a software module.

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.

/ 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.

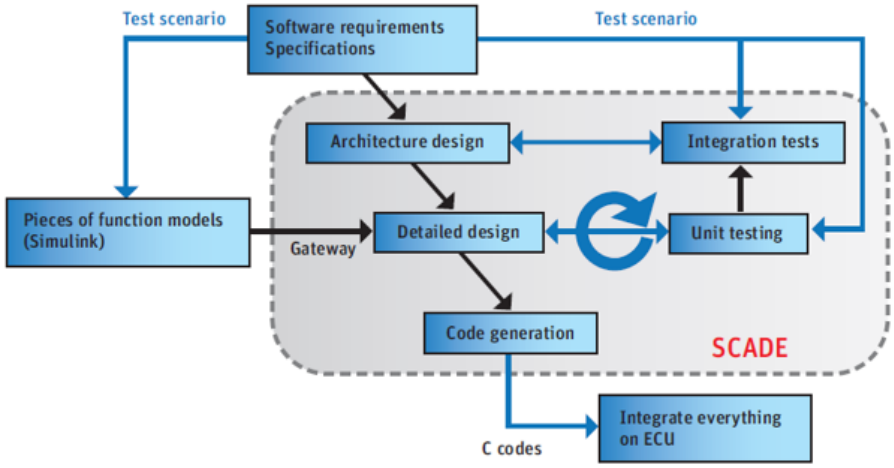


Figure 12: HEV software development process with SCADE at Suburu.

/ 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. Ansys has the only comprehensive simulation solution to produce ISO 26262 certifiable code for autonomous vehicles, ensuring the safety of autonomous vehicles. This complete toolset allows you 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.

/ 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

/ 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

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

If you've ever seen a rocket launch, flown on an airplane, driven a car, used a computer, touched a mobile device, crossed a bridge or put on wearable technology, chances are you've used a product where Ansys software played a critical role in its creation. Ansys is the global leader in engineering simulation. We help the world's most innovative companies deliver radically better products to their customers. By offering the best and broadest portfolio of engineering simulation software, we help them solve the most complex design challenges and engineer products limited only by imagination.

Visit www.ansys.com for more information.