ZVEI:
Die Elektroindustrie

Best Practice Guideline

Software for Safety-Related Automotive Systems

German Electrical and Electronic Manufacturers' Association

Imprint
Best Practice Guideline
Software for Safety-Related Automotive Systems

Publisher:
ZVEI - German Electrical and Electronic
Manufacturers' Association
Automotive – Electronics, Infrastructure & Software
Lyoner Strasse 9
60528 Frankfurt am Main, Germany

Phone: +49 69 6302-276
Fax: +49 69 6302-407
E-mail: zvei-be@zvei.org
www.zvei.org

Responsible: Dr. Stefan Gutschling

Dezember 2016

Table of Contents

1. Objectives of this Guideline

This guideline provides lessons-learned, experiences and best practices related to the application of ISO 26262 for the development of software. Please note that the guidelines given are of general nature and do not replace a thorough consideration of the project specific development regarding achievement of "Functional Safety" considering ISO 26262.

2. Overview

This guideline is intended to be maintained and extended. The current version addresses the following aspects:

- Definition of terms used in the context of "Functional Safety" and software development.
- Guidance for safety concepts and architectures for safety-related software.
- Classification and qualification of software tools used in the development of embedded software.

3. Explanation of Terms

The following explanations include terms used in this document. The explanations are intended to ease the common understanding.

Term	Description
QM software	Software that is not developed according to ISO 26262 ASIL A, to D but still the software is developed according a well-defined process (e. g. an ASPICE compliant process). QM software must not be used to realize safety-related functionalities and special consideration is needed if QM software is integrated in an ECU that realizes safety-related functionalities.
Silent software	"Silent software" is a term used to describe software that does not interfere with other software with respect to memory access (e. g. range-check of index values, verification of pointer access) under the conditions defined in the Safety Manual. "Silent" software does not fulfill specific safety-related functions.
Implicitly safe	"Implicitly safe" is a term used to describe software that is silent software with additional dedicated timing properties (e. g. with respect to execution time, deadlocks and robustness with respect to input signals) under the conditions defined in the Safety Manual. "Implicitly safe" software does not fulfill specific safety-related functions.
Safety manual	A Safety Manual describes constraints and required activities for the integration and/or usage of elements that have been developed and prequalified acc. ISO 26262 as Safety Element out of Context.
Safe, safety, explicitly safe	"Safety/safe/explicitly safe software" is a term used to describe software that fulfills specific safety-related requirements under the conditions stated in the safety manual.
"Trusted mode", system mode, privileged mode, supervisor mode	CPU mode for executing software with full access to the hardware features of the microcontroller. Software executed in this mode poses a higher risk and should be treated as such (e. g. development according to required ASIL including the implementation of appropriate safety measures).
Safety-related functionality	A functionality that realizes safety requirements.

Table 1: Explanations of terms used in this document

4. Software Safety Concepts and Architectures

4.1. Introduction

In this section different software safety concepts are depicted and some hints are given to decide for the appropriate safety concept depending on the conditions in a specific development project.

Often many of the functionalities and properties of ECU software are not safety-related, but only a part of them. Only those software elements that contribute to the implementation of safety requirements are considered safety-related.

To implement a mix of safety-related and non-safety-related functionalities there are two fundamental design options mentioned in ISO 26262:

- Develop a design in which such a mix can coexist. This is often called "**Mixed ASIL Design**" and is a typical approach if the portion of safety-related functionalities is rather small or third-party or QM software needs to be integrated.

or

- Develop the complete ECU software in conformance with the "Maximum ASIL" assigned to any of the safety-related functions within the ECU. This is often called "**Maximum ASIL Design**" and the typical approach if the portion of safety-related functionalities is rather large.

Figure 1 depicts different ECU types holding software elements with and without related safety requirements and illustrates these two design patterns.

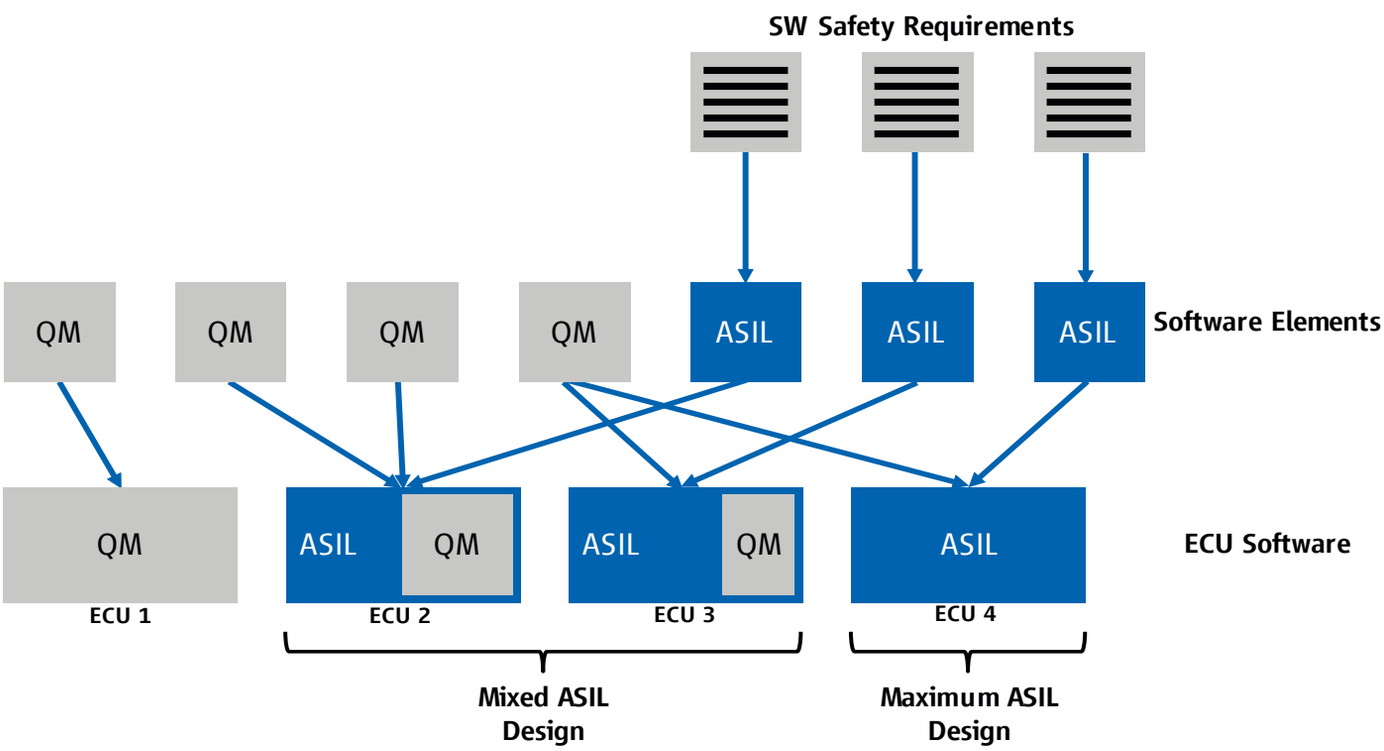

Figure 1: Mapping of software safety requirements to ECUs – Source: ZVEI

Both design options must focus on the same goal: To achieve the necessary integrity of the safety functions. The level of integrity expresses the degree of trust you can have that a software will provide the stated functions and properties as demanded under specified conditions.

Necessary integrity can be achieved in two ways: One is to prevent that the software contains errors which lead to a malfunctioning behavior. Another is to include technical measures that are able to detect and control such a malfunctioning behavior.

In a "Mixed ASIL Design" the elements do not all have the same integrity based on their specific development goals. If they are integrated into one software without further measures, the integrity of the complete software cannot exceed that of the element with the lowest integrity, like the weakest link of a chain.

To achieve a higher degree of overall integrity one must provide evidence that the elements with a lower integrity are not able to interfere with the elements of the target ASIL which is called "achieving Freedom from Interference". There are two principles to argue "Freedom from Interference":

- Detect that an interference has occurred and mitigate the effects
- Prevent that an interference occurs

Detection and mitigation is sufficient if the resulting (degraded) functional behavior of the software can still ensure "Functional Safety" (e. g. achieve and maintain a safe state).

In a "Maximum ASIL Design" all elements have the same integrity. When integrating such elements, in principle the complete software has the same integrity and does not require an examination for Freedom from Interference. Nevertheless, the safety analysis at software architectural level may reveal weaknesses which have to be addressed (e. g. by technical measures) in order to achieve confidence in "Functional Safety".

The following sections describe the two approaches in further detail. Since software architectures according to AUTOSAR are more and more used it is mentioned which contribution AUTOSAR features could provide.

4.2 "Mixed ASIL Design"

A "Mixed ASIL Design" targets the development of software elements according to QM or a lower ASIL without jeopardizing the integrity of the entire software system, which may have a higher ASIL. It may also enable the containment of errors in a partition.

This concept requires a suitable software design on application level, i. e. functional blocks must be coherent and unwanted interlinking between functional blocks (e. g. via global variables) should be avoided. It also requires a safety mechanism realizing the freedom from interference on hardware and software level which ensures that a software element with a lower ASIL cannot interfere with a software element with a higher ASIL. This mechanism must be able to either prevent that a malfunction of one element leads to the malfunction of another element, or it must be able to detect such interference and to mitigate the effects in time. This safety mechanism has to be developed according to the "Maximum ASIL" of the software safety requirements realized on this ECU.

ISO 26262 mentions different aspects of possible interferences:

1. Memory, which includes the RAM as well as the CPU registers
2. Timing and executions, which refers to blocking of execution, deadlocks and livelocks or the incorrect allocation of execution time in general
3. Communication, summarizing all possible errors that could occur in the communication between software elements both within the ECU and across ECU boundaries.

The separation between "QM or lower ASIL" and "Maximum ASIL" elements provides the following benefits:

- Development methods for "Maximum ASIL" only have to be applied for safety-related software elements (which includes the elements ensuring the freedom from interference). This allows the reuse of existing QM software (e. g. third-party software), as long as it is not safety-related.
- Propagation of failures between software

elements of the same ASIL can be prevented or detected, although it is not mandated by Freedom from Interference. However this also supports the separation of safety-related parts with high availability requirements from other parts in fail-operational architectures.
- Some failures caused by hardware defects can also be prevented or detected (e. g. timing supervision will detect a faulty clock source).

On the other hand, the following disadvantages have to be taken into account when applying the "Mixed ASIL Design":
- The separation adds additional complexity to the software design. Especially in legacy software safety-related and non-safety-related functional blocks are often tightly coupled, which requires additional effort for a software architecture redesign.
- The safety mechanism to ensure "Freedom from interference" may result in a performance penalty during runtime (e. g. for reprogramming the MPU and context switching). To reduce these penalties to a minimum, the interaction between the software elements that are separated by freedom from interference mechanisms needs to be as low as possible.

4.3 "Maximum ASIL Design"

The "Maximum ASIL Design" has its advantages in use cases where a high share of the software provides safety-related functionality. In this approach, both the safety-related and the non-safety-related functions follow the development process of the highest ASIL in the system. For the non-safety-related software elements, the coexistence argumentation follows a process argumentation: if those software elements are developed in the same stringent way applying the same process methods as the safety-related software elements, the coexistence of the elements is possible without further technical separation measures. The only difference between the non-safety-related and the safety-related software elements is then the necessary safety analysis for the latter.

Compared to the "Mixed ASIL Design" this approach gives the following benefits:
- No additional complexity for development of a partitioning concept.
- No performance penalty due to safety mechanisms ensuring Freedom from Interference.
- Improved quality also for the non-safety-related software components which leads to a higher availability of the system.

On the other hand the following disadvantages have to be considered:
- The development effort increases since all software elements have to be developed according to the highest ASIL. For the non-safety-related part an additional safety requirement is then applied, which requires the non-interference ("silence") with the safety-related part.
- As ASIL development does not mean that the software is error free, errors in these parts are not prevented to propagate by design.
- Inclusion of third-party software (e. g. "black-box" software) is more difficult, as the development process of these modules is often unknown or cannot be influenced.

4.4 Mechanisms to realize freedom from interference

The following paragraphs contain suggested protection mechanisms for different kinds of fault classes in the data and control flow domain, which includes faults listed in annex D of ISO 26262 part 6. Data faults are either related to global data, to data residing on the execution stack, or to data received by QM SWCs. Additionally, hardware register faults constitute a special kind of data faults. Control flow faults are either related to timing faults or to interrupt faults. Faults due to illegal references can have an effect on either the data or the control flow domain.

Please note: The following list includes mechanisms sufficient for typical ASIL A or B projects but it also shows additional mechanisms that can also be used for higher ASILs. Especially those mechanisms required for higher ASILs are typically supported by AUTOSAR Basic Software features.

Fault class: "Global Data Faults"

There are several options to address this fault class:

1. By partitioning the available RAM memory space in QM and ASIL parts and cyclically verifying a memory marker in between (initialized to a specific pattern), the probability to detect a relevant buffer overflow originating in QM software is increased.
2. To protect safety-related data without using an MPU, double inverse storage concepts can be employed to detect accidental overwrites by QM software by comparing the original variables to bit-inverse shadow copies upon reading or cyclically (as long as the fault tolerance time is considered). If a larger set of data is not written frequently, memory-efficient checksums can be used to detect accidental modifications of data parts. This protects against QM data pointer corruptions and QM buffer overflows, both resulting in writes to ASIL data.
3. To protect against accidental overwrites the CPU's memory protection unit (MPU) can be used together with an allocation of tasks to separate partitions. In AUTOSAR, it is the responsibility of the Operating System to handle the MPU and thereby to ensure a proper separation between the entities. This is typically required for ASIL C and D but can also be useful or even required for lower ASILs.

Fault class: "Stack Faults"

There are several options to address this fault class:

1. By using a stack range check that checks whether the current stack pointer is in range of the allocated stack memory, the probability to detect a stack overflow or underflow by QM software modifying the stack pointer can be increased. Such a stack check can be implemented cyclically or – in most cases even better – in context of a task switch.
2. Additionally, stack overflows and underflows can be detected by checking memory markers (initialized to a specific pattern) placed above and below the allocated stack memory, which detects a subset of stack faults. This feature is also part of the AUTOSAR Operating System. Please be aware that this mechanism cannot detect stack overflows that do not overwrite the memory markers.
3. The stack can also be protected by a hardware MPU which actually prevents all stack faults. This is typically required for ASIL C and D but can also be useful or even required for lower ASILs.

Fault class: "Less Reliable QM Data Quality"

If data that is relevant to safety-related ASIL calculations is routed through QM software parts (e. g., drivers or communication stacks that process hardware input) that could corrupt data, there are several options to address this:

1. A single sporadic fault can be detected via a plausibility check. Such a plausibility check can use either values from other sources or previous values from the same source as an additional input. For instance, receiving a speed value of 0 km/h after having received one of 100 km/h in the previous CAN message 20 ms before is not plausible. Please note that the detection probability depends strongly on the assumed fault model.
2. Alternatively and with a higher detection probability, end to end protection checksums and signal alive checks can be used. The AUTOSAR end-to-end protection modules have been specified for this purpose.

Fault class: "Hardware Register Faults"

To protect against QM software parts accidentally modifying hardware register state that is safety-related, there are several options:

1. Some microcontrollers offer locks for selected configuration registers or configurable write-once semantics, which should be used.
2. A cyclic check of the current hardware state against the expected state as held in software can be performed to detect faults as long as the fault tolerance time is considered.
3. Use a pro-active recovery mechanism that periodically rewrites the expected register states (assuming single bit flips as fault model).
4. The strongest mechanism is the protection of memory mapped registers via the MPU. Some CPUs also provide a Peripheral Protection Unit for this task. This is typically required for ASIL C and D but can also be useful or even required for lower ASILs.

Fault class: "Timing and Execution Faults"

To protect against QM software significantly delaying or even blocking ASIL software execution, there are several options:

1. Hardware or software watchdogs can be used. These should either be configured in a window mode, or they should regularly be triggered at the end of its deadline to detect delays as early as possible.
2. Depending on the scheduling scheme employed in the basic software operating system, overflows of time slices or task overruns can be detected. This is also a feature of the AUTOSAR Operating System.
3. The strongest mechanism that also detects fault in the program logic is the supervision of the program flow in combination with time stamps. This is also a feature of the AUTOSAR Watchdog Stack and is typically needed only for ASIL C and D.

Fault Class: "Interrupt Faults"

To protect against the fault that global interrupts or ASIL interrupt sources are permanently disabled by QM software parts, both properties can be checked cyclically to be enabled in an assertion.

To protect against QM **I**nterrupt **S**ervice **R**outines executing at higher rate than expected, which will delay or even block the execution of ASIL ISRs, two measures can be taken:

1. If possible from the real-time scheduling point of view, ASIL ISRs should be given a higher priority compared to QM ISRs.
2. As a monitoring measure, the arrival rate of QM ISRs can be monitored to be in range of the expected rate. This is also a feature of the AUTOSAR Operating System.

Fault class: "Illegal References"

By referencing ASIL symbols, QM software could include code that writes to protected ASIL data or executes protected ASIL functions. This misbehavior can be protected against by partitioning the software in the design phase. By explicitly denoting ASIL data and function declarations that are legal to be referenced from within QM software parts in an ASIL/QM interface header, this design by contract can be proven in an automated way. An example approach would be to implement the interface header in a dummy module and link it to the QM software parts. The linker will then report undefined references from QM to ASIL software parts, which states an illegal interference. This proof is especially important when integrating QM third-party code, and the explicit interface can additionally be used to integrate plausibility checks when transitioning from/to QM software (see also fault class "less reliable QM data quality").

5. Confidence in the Use of Software Tools

5.1. Motivation

Software tools play a major role in the implementation of processes and methods used during the development of safety-related systems, software and hardware.

Using tools can be beneficial because they enable, support or automate safety-related development activities (e. g. development and management of requirements or architectural designs, code generation, analyses, testing or configuration management).

However, in case of a malfunctioning behavior such tools may also have adverse effects on the results of tool-supported development activities and thus on the "Functional Safety" achieved in the final product or its elements including software.

ISO 26262 provides an approach to achieve confidence that using software tools does not jeopardize "Functional Safety". This approach contains:

- Determination of single tools or tool chains which are relevant for safety-related activities and identification of the used functionalities and their purpose during development.
- An analysis to determine the required confidence for each relevant software tool, based on the risks related to the used functionalities and its role in the development process ("classification").
- Measures to qualify a software tool, if the classification indicates that this additional risk reduction is needed.

This approach can be supported by the tool vendor, e. g. by providing information such as generic analyses based on intended application use cases or test cases and test suites for tool qualification. The responsibility for using the tool in a suitable way remains with the user.

The following sections describe this approach in further detail.

5.2. Analysis and classification of software tools

The risk related to the tool functionalities used for a specific purpose during development is determined by the tool´s impact and the possibility to detect malfunctions yielding the aggregated tool confidence level (TCL):

1. The tool impact (TI) expresses the possibility that a malfunction of a particular software tool can introduce or fail to detect errors in a safety-related item or element being developed.
   - TI1: Shall be selected when there is an argument that there is no such possibility
   - TI2: Shall be selected in all other cases
2. The tool error detection (TD) expresses the confidence that due to tool-internal or tool-external measures (e. g. subsequent process activities) relevant tool malfunctions producing erroneous output can be prevented or detected.
   - TD1: High degree of confidence (that a malfunction and its corresponding erroneous output will be prevented or detected)
   - TD2, TD3: Medium or low degree of confidence

The classification depends on the usage of the tool (e. g. used functionalities) as part of the complete development process.

Figure 2 shows the approach and table 1 gives some examples. Please note that the specific workflow embedding the tool usage has to be considered.

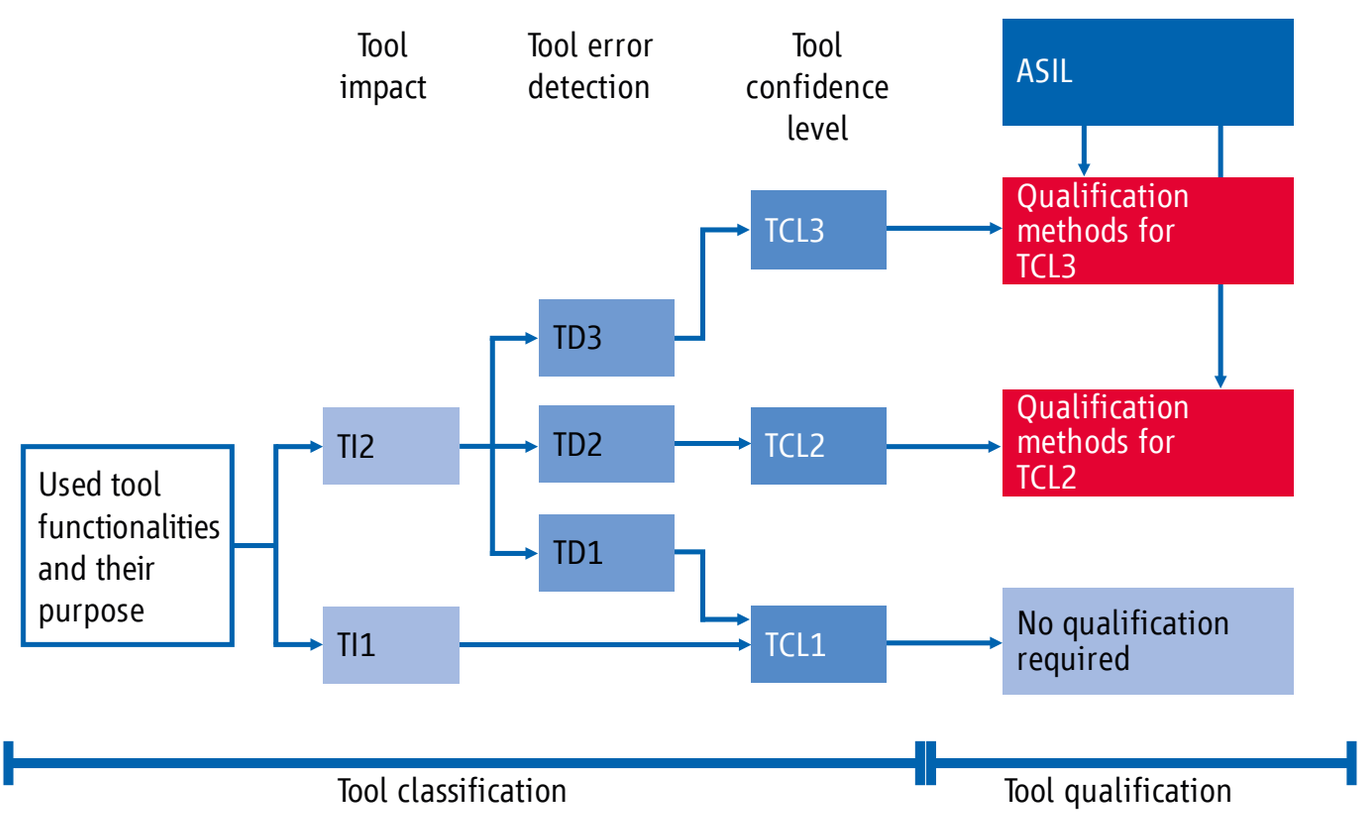

Figure 2: Classification and qualification of software tools acc. ISO 26262 – Source: ZVEI

Tool	Use case	Failure mode	TI	Measures to detect or prevent malfunctioning of tool	TD	Rationale	TCL	Qualification needed
C-Code generator	Generate C-Code from model	Incorrect translation from model to code	TI2	None	TD3	Errors are not detected if no systematic tests are performed.	TCL3	Yes (TCL3)
				Full verification of code with required coverage by tests, reviews and static code analysis	TD1	Errors are detected by verification.	TCL1	No
				Full verification of code with code generator specific checker tool	TD1	Errors are detected by checker tool.	TCL1	No
				Use redundant code generator and compare results	TD1	Failure of one tools will be detected by the other tool. Equal failure of both tools is unlikely	TCL1	No
Static code analysis tool	Static code analysis	False negatives with respect to specified error class (e. g. array out of bounds for a bounds checking tool)	TI2	None	TD3	Other tests do not focus on this error class	TCL3	Yes (TCL3)
Configuration management tool	Checkout specific artifact version	Checkout of wrong artifact version	TI2	Artifact checksum verified against external database	TD1	Corrupted data and wrong artifact version will be detected externally	TCL1	No
		Artifact was corrupted	TI2	Artifact checksum verified against tool internal database	TD1	Corrupted data will be detected internally	TCL1	No

Table 2: Examples for tool classification

5.3 Qualification of software tools

The resulting TCL may be reduced by improving the detection or avoidance measures (iterative tool analysis). As a consequence alterations in the process (e. g. removal of a redundant tool in the tool chain) may invalidate the TCL argumentation.

Example: If an analysis shows that for the tool and its intended usage a TCL1 cannot be argued, there are at least two options:

- Lowering the TCL by improving the TD introducing additional detection or prevention measures into the development process (e. g. checking tool outputs) or into the tool itself.
- Performing a qualification of the tool according to the TCL for the target ASIL if lowering the TCL is not feasible or not efficient.

The quality of the documentation and the granularity of the tool analysis require an adequate level of detail so that the resulting TCL is comprehensible and the resulting TCL can be justified (Neither a very detailed investigation nor a rough general view is helpful).

For TCL1 classified software tools no qualification measures are required at all.

For TCL2 and TCL3, tool qualification measures provide evidence that justifies confidence in a software tool for its intended use cases in the development environment. The following measures are applicable depending on the TCL and target ASIL:

- Increased confidence from use.
- Evidence for a structured tool development process.
- Tool development in compliance with a safety standard.
- Validation of the software tool.

6. Participating Companies

Participating companies in the “UG Software ISO 26262” working group

Automotive Lighting Reutlingen GmbH

Brose Fahrzeugteile GmbH & Co. KG

Elektrobit Automotive GmbH

Elmos Semiconductor AG

Infineon Technologies AG

Kugler Maag CIE GmbH

Lear Corporation GmbH, Werk Kronach

Marquardt GmbH

Mahle International GmbH

nova motum Services & Consulting GmbH

NXP Semiconductors Germany GmbH

OptE GP Consulting Optimize E Global Performance

Qestronic GmbH

Sanmina-SCI Germany GmbH

STMicroelectronics Application GmbH

TTTech Automotive GmbH

Vector Informatik GmbH

Webasto Thermo & Comfort SE

ZVEI - German Electrical and Electronic
Manufacturers' Association
Lyoner Strasse 9
60528 Frankfurt am Main, Germany

Phone: +49 69 6302-0
Fax: +49 69 6302-317
E-mail: zvei@zvei.org
www.zvei.org

Source: XXX