ISO 26262 compliant verification of functional requirements in the model-based software development process

Hans J. Holberg
SVP Marketing & Sales, BTC Embedded Systems AG
An der Schmiede 4, 26135 Oldenburg, Germany
hans.j.holberg@btc-es.de

Dr. Udo Brockmeyer
CEO, BTC Embedded Systems AG
An der Schmiede 4, 26135 Oldenburg, Germany
udo.brockmeyer@btc-es.de

Abstract: The model-based software development process is generally accepted in the automotive and aerospace domain. More or less seamless tool-chains support the model-based approach in order to help improving the functional safety aspects of such processes while keeping the efficiency under growing complexity concerns. In the last decade, fundamental progress and improvements in the area of modelling, simulation and automatic code generation have been achieved. Even in the area of fully automated structural testing, various solutions have been successfully entered the tool-chains. Concerning the verification of functional requirements in the model-based domain, there is still big room for improvements. Especially the demand of having an automatic, scalable approach for functional testing and formal verification is not yet achieved. This paper presents an automatic approach that has been developed in order to efficiently support international standards regarding functional safety, like ISO 26262 for automotive. It presents an integrated method to use automatically synthesized C-code observer fragments from formalized specifications. Then requirements based functional test and formal verification can be almost automated as the synthesised C-code observers are automatically embedded into a test and verification tool environment. This includes the model, code and object code levels such that a very general use of C-observers can be shown. The automation of this approach includes the requirements-based test case generation, automatic test execution and analysis, as well as test quality measurement and coverage of requirements. The described method effectively and smoothly fits into the framework of software quality standards as it is for instance specified in the new automotive standard for functional safety ISO 26262. The approach has already been implemented in a first version for the Matlab/Simulink tool chain on top of the production code generator TargetLink from dSPACE. Further future potential of such observer technology, for instance 'embedded diagnostics' by using C-observers, will also be discussed.

1 Field of Application

The presented method[1] has been introduced within a widely used model and auto code[2] based testing and verification tool environment[3] as an extension[4] to enable automatic requirements based testing. The testing tool environment's main use case in the past was the automatic structural back-to-back testing between MiL[5], SiL[6] and PiL[7] including full automatic structural test vector generation to ensure a maximum model and code coverage up to MC/DC[8]. This approach allows to automatically test all development steps from the model level down to the implementation level. It finally lifts testing up to the model level, hence called model based testing. The model based approach is in the main focus of this development and test environment, but even any kind of C-Code resulting from other code generation and even hand written code sources is supported. The main use case structural back-to-back testing is supporting the recommended ISO 26262 methodology and has been certified by an independent certification body to be "suitable for purpose" for all defined ASILs[9] from A to D. This touches an important described method of the ISO 26262, but requirements related testing methodologies of the ISO 26262 are not automated by this back-to-back testing approach. In order to enable more automatic testing for requirement-based testing of the testing tool environment an extension to the former introduced main use case is described in the following chapters. Two main topics are addressed. First, the new extension shall cover all important recommendations of the ISO 26262 concerning requirements based testing, and second it shall automate the testing as much as possible within the MBD[10] Process.

This new approach has been introduced successfully in the automotive domain last year in Germany and Japan. First results are very promising regarding three aspects:

- Smooth integration in the existing testing process
- Efficiency gains
- Quality improvements

[1] This work has been partially funded by ARTEMIS in the European project CESAR.

[2] Here: Matlab Simulink/Stateflow (The Mathworks) in combination with the leading automotive code generator TargetLink (dSPACE GmbH) has been used in real serial production projects as standard modelling and code generation environment.

[3] Here: BTC EmbeddedTester from BTC Embedded Systems AG is used. It became a standard test and verification tool environment for TargetLink users in the automotive domain.

[4] BTC EmbeddedTester extension: Requirement-based Verification and Testing with C-Observer

[5] MiL: Model in the Loop. Normally it is a closed-loop system model which consists of the control component plus plant (environmental) model. Here, an open-loop with an automatically generated test harness is used to automatically test the SUT (System under Test).

[6] SiL: Software in the Loop. In contrast to PiL, the real target hardware is replaced by the used host-computer and its ordinary processor. The developed model of the software is only translated into target hardware compatible code. The plant model is replaced by a test driver (automatically generated test harness).

[7] PiL: Processor in the Loop. In contrast to SiL real target hardware (evaluation board) is used to load the application on it for testing. This allows identifying compiler- and processor issues.

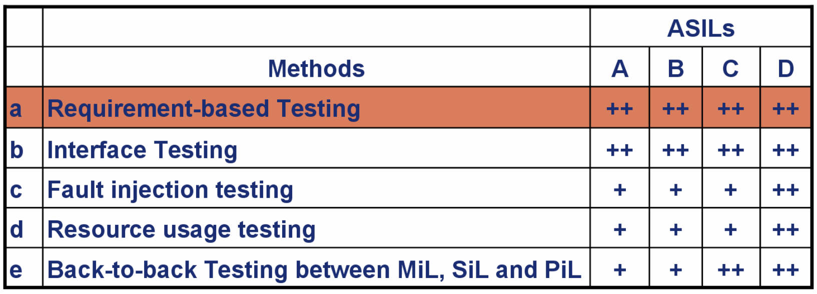
[8] Modified Condition Decision Coverage (MC/DC): A set of test vectors, which make every decision TRUE and False while each single condition of that decisions has an independent influence on the value of that decision. A 100% MC/DC coverage guarantees the detection of any failure within a decision of the mode or model.

[9] Automotive Safety Integrity Levels (ASIL A, ASIL B, ASIL C and ASIL D). Level A is the lowest and D the highest safety integrity level.

[10] Model Based Development

2 ISO 26262 Software Testing and Verification Tasks

As the upcoming[11] automotive standard ISO 26262 is one of the most important state of the art functional safety foundation for any testing and verification tasks in this industry domain, the relevant definitions and recommendations regarding requirements based testing are summarized within this chapter. The following figure shows the importance of the Requirement-based Testing[12] for all ASILs in the ISO 26262.



		ASILs			
	Methods	A	B	C	D
a	Requirement-based Testing	++	++	++	++
b	Interface Testing	++	++	++	++
c	Fault injection testing	+	+	+	++
d	Resource usage testing	+	+	+	++
e	Back-to-back Testing between MiL, SiL and PiL	+	+	++	++

Figure 1: ISO 26262 recommendations regarding Software Unit Testing

The ISO 26262 distinguishes three kinds of requirement notions:

- Informal Notations.
  This description technique does **not** have its syntax defined completely. This kind of documentation is widely used in an intuitive way; for example natural language/ informal text definitions of requirements and any kind of figures and drawings.

- Semi-formal Notions.
  If the syntax of a notation is completely defined, but the semantics definition is incomplete, it is called semi-formal. It is an example of a machine readable specification, which can not be used for any further analysis. One example is an UML Use-Case diagram, which has its syntax and ambiguous interpretations.

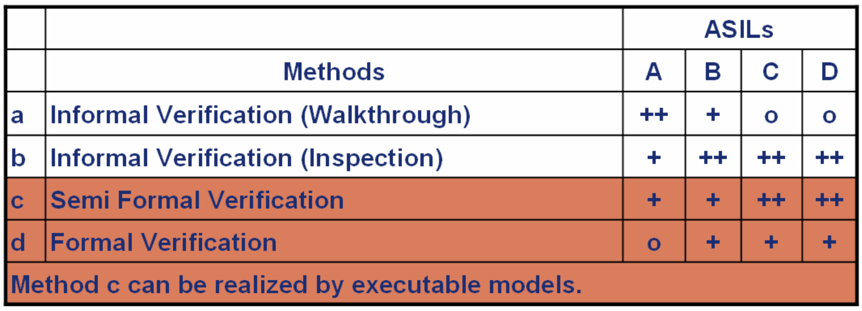
- Formal Notations.
  This describes a technique that has both, its syntax and semantics defined completely. Example for this are executable models, c-code and formal language specifications.

In order to automate any kind of testing based on requirements, it is obvious to use Formal Notations, which are machine readable and which can be used for further algorithmic analysis techniques. Trade-off on the other side is the difficulty for a human being to fill the gap between informal and formal specification. This problem will be

[11] The final release of this international standard is planed for 2011 and it is available as a Draft International Standard ("DIS") since mid of 2009.
[12] ++ means „Highly Recommended“

addressed in one of the following chapters by introducing an intuitive high level abstraction specification method called Pattern Approach.

A Formal Specification in the sense of ISO 26262 is defined as a method which is based on a specific Formal Notation. This formal specification part is addressed by the described method upon so-called C-Observer Specification, with syntax and semantics is well defined. The relationship between the high-level user friendly formal requirement specification level and the technical formal realization will be defined later in this paper.



		ASILs			
	Methods	A	B	C	D
a	Informal Verification (Walkthrough)	++	+	o	o
b	Informal Verification (Inspection)	+	++	++	++
c	Semi Formal Verification	+	+	++	++
d	Formal Verification	o	+	+	+
Method c can be realized by executable models.					

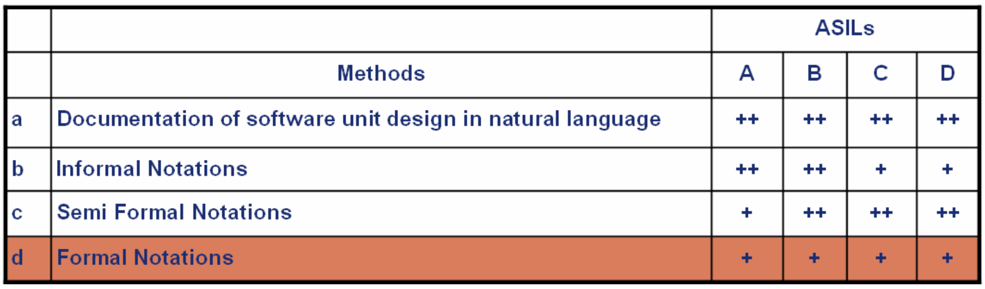
Figure 2: ISO 26262 recommendations regarding verification of requirements

Semi-formal and Formal Verification plays an important role as methods for the verification of requirements of ASILs B to D, as can be seen in the table above. It is also of interest especially regarding automatic approaches, that Semi-formal Verification can be fulfilled by executable models. In other words, it can be done via model simulation.

Formal Verification on the other hand is defined as a method which is used to ensure the correctness of an SUT[13] against a Formal Specification of its required behavior. The standard is **not** talking about Formal Verification as a complete mathematical method. It is defining Formal Verification simply upon Formal Specification of Requirements as a basis for the verification task. Thus, any testing activity, which is done on the basis of clear syntactical and semantical specifications, is fulfilling the ISO 26262 recommendations in relation to Formal Verification. The ISO standard recommends the introduction of quality measurement and coverage metrics to fulfill certain ASILs, for instance Formal Verification. It defines a maturity gate regarding requirements by introducing the term **Requirements Coverage** defined in a more intuitive way.

In order to use this important maturity gate in the context of automatic testing, a new approach of measuring Requirements Coverage is introduced within the described method. It will be defined later on with C-Observer Code Coverage.

[13] System Under Test („SUT“)



		ASILs			
	Methods	A	B	C	D
a	Documentation of software unit design in natural language	++	++	++	++
b	Informal Notations	++	++	+	+
c	Semi Formal Notations	+	++	++	++
d	Formal Notations	+	+	+	+

Figure 3: ISO 26262 recommendations regarding notations of unit designs

When an MBD process is used, which introduces executable models, the SUT has a machine readable and unique interpretation, which is the preliminary for automatic testing approaches. The figure above shows that Formal Notations are recommended (+) for all ASILs.

The next chapter will combine the ISO 26262 derived recommended methods:

- Executable Model (Formal Notations of Designs)
- Specification and Verification Approach (Formal Verification)
- Quality Measurement of the Verification and Test Activities (Coverage)

together with existing automatic test/verification and formal specification technologies in order to extend the existing back-to-back testing approach by automatic requirement-based testing to allow ISO 26262 compliant MBD.

3 C-Code Observer Concept embedded in the Virtual Verification Platform

The VVP[14] is used as a semantical basis for any kind of analysis techniques. In this case, the behavioural description of the SUT, the environment of the SUT and the requirements were given as C-Code within the VVP-Architecture which can be seen in the figure further done.

C-Code as a semantical basis for test- and verification activities has a lot of advantages in practice as C-code is a de facto standard in the development of embedded systems in the automotive domain. Hence any given C-Code of the SUT or the Environment Specification can be re-used in this approach.

The base technology of the existing testing environment works on self-contained C-Code in order to automatically analyse the SUT regarding any given test- and check property. Besides the C-Code semantical basis, the interfaces are important for any analysis like automatic test case generation.

[14] Virtual Verification Platform („VVP“). It is needed to have a clear machine readable specification of the test platform in which the SUT will be checked. It can be used as execution platform or/and as input for any automatic test- and verification technologies like Model Checking Engines, Test Vector Generation engines, structural analysis engines to find standard errors like data overflow, loop-divergence, dead-code etc.

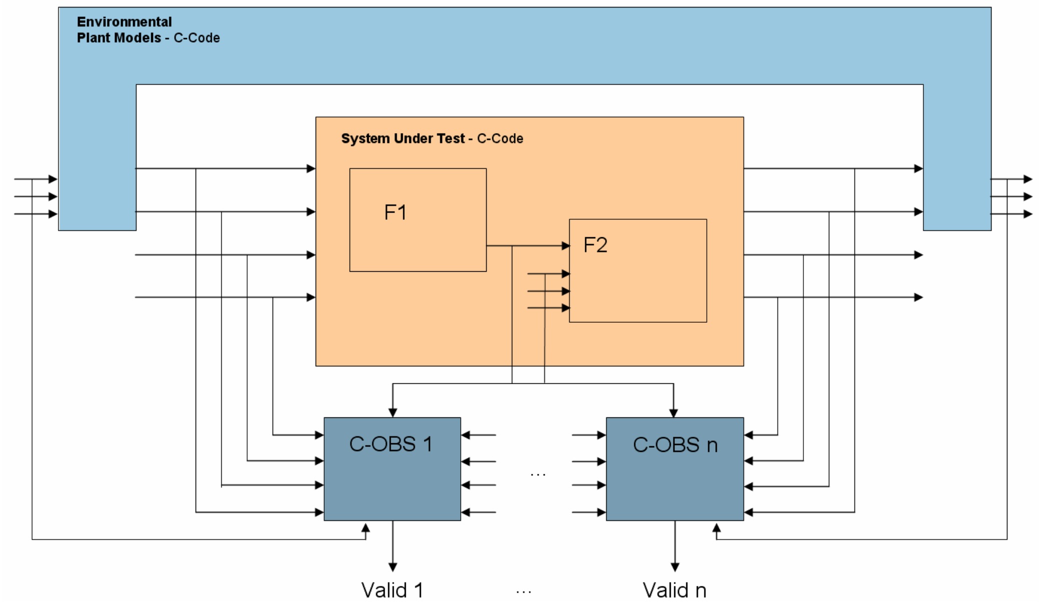


Figure 4: Virtual Verification Platform enhanced by C-Code-Observers

The SUT with its software architecture (functions and its wiring) is given as self-contained C-Code automatically generated by an auto code generator of the functional or implementation model. The environment of the SUT is also given as C-Code, which can be reused from any plant model descriptions or can be synthesized from given environmental high-level specifications. The possibility of synthesizing environment assumption from high level specification languages is discussed later in this paper.

The SUT corresponding requirements are represented by so-called C-Observers (C-OBS$_{1..n}$). These observers are in general small C-functions running in parallel to the SUT during any test or analysis step in order to observe the correctness of the behavior of the SUT in respect to the described requirements. The C-Observer Functions return so-called valid-signals (Valid$_{1..n}$), which indicate accepted behavior with a TRUE (1) or error states with a FALSE (0). This allows automation of the test validation, if the requirements are completely represented by such observers. The next chapter is showing the bridge between Informal Requirements and the needed C-Observers which are incorporated in the VVP.

4 Connecting Requirements to C-Observers

In order to bring the requirements into the VVP, a well linked information chain has to be established (see next figure). Given an Informal Requirement specification in the beginning, no concrete information about the final implementation or the design is known yet. Normally an Informal Requirement is represented in natural language. This has to be refined step-by-step along the MBD process. The refinement is performed until a Formal Requirement Specification is available by using different methods in the automotive industry. It depends on the specific application class and on the existing user processes.

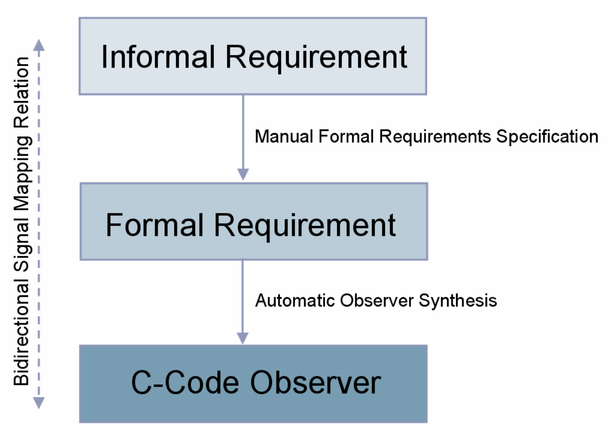

Informal Requirement
Manual Formal Requirements Specification
Formal Requirement
Automatic Observer Synthesis
C-Code Observer
Bidirectional Signal Mapping Relation

Figure 5: Way from Informal Specification down to C-Code-Observers

In our described testing and verification tool environment, users are leveraging from the Pattern Specification Approach from an early specification stage on in order to perform the manual formal specification task. This approach provides a library of predefined patterns for specifying functional (safety- and mission critical) requirements. Patterns can be instantiated simply by filling the pattern parameters with Boolean expressions ranging over model elements. The pattern specification method guarantees an easy user entry in the formal world, without having a deep mathematical and theoretical background. This schematic pattern approach allows full certainty about what has been formally specified, without any final doubt. If this has been done accurately, a synthesis algorithm can generate the C-Code-Observers fully automatically and efficiently from the Pattern Specification. Beside the three explained specification stages, a bidirectional mapping table is managed fully automatically to ensure full traceability from textual events to model signals down to C-Code variables. The correspondence between model elements and code variables is provided by the modelling tool and the auto code generator.

5 Environmental and Assumption Specification via C-Observers/C-Drivers

A requirement in general consists of an assumption and a commitment part. A set of assumptions shall define the needed environmental behavior of an SUT to make a specific commitment a valid requirement specification. This is called the Virtual Integration of the SUT in VVP.

The following formula shall describe the mathematical relationship between the set of Assumptions (A_1 … A_n) and the commitment C, which shall be granted by the SUT:

$$A_1 \text{ and } A_2 \text{ and } \dots \text{ and } A_n \Rightarrow C$$

As C-Observers are used to represent the Commitment C, also the Assumptions (A_1 … A_n) can be represented via C-Observers. There are two different possible modes of assumption handling in the virtual verification platform. Either the pure acceptance of the assumptions can be observed or the strict compliance may be forced by the environment part of the VVP.

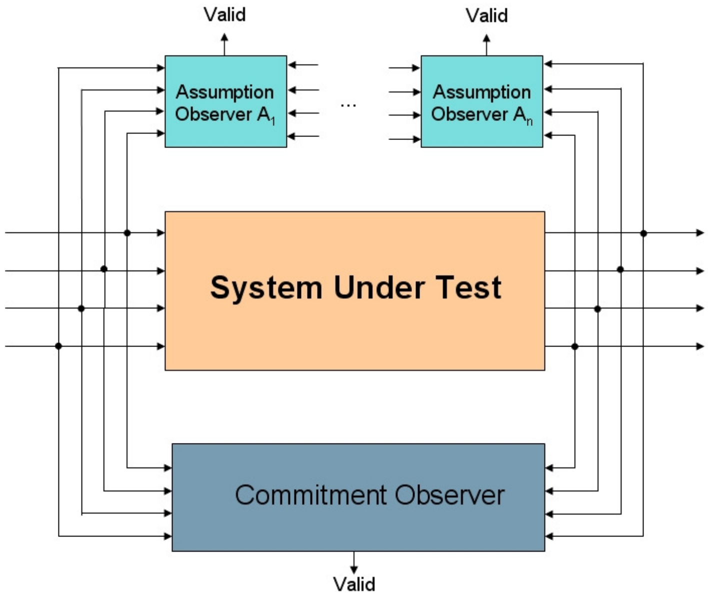


Figure 6: Assumption Observers virtually integrated in the VVP
to evaluate valid test runs for commitment checks

The first possibility of the pure acceptance can be realized easily with pure observers as discussed on the commitment part before.

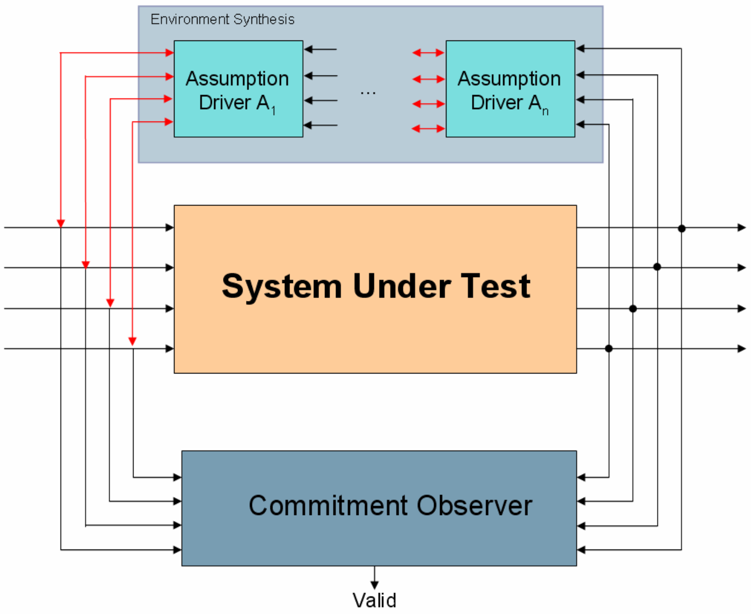


Figure 7: Assumption Drivers virtually integrated in the VVP
with direct influence on input interface of SUT (red arrows)

The second possibility requires that the observers are converted into drivers. Those drivers are able to change input signals of the SUT in dependency to the SUT output behavior. In general, single assumption drivers will have an influence on only parts of the input-interface, while the rest is driven by other assumption drivers or by free inputs driven by test vectors. Assumption Driver signal writings on the input interface of the SUT have priority over test vector stimuli test execution.

It is obvious, that this concept of parallel composition of assumption drivers could cause trouble, as multiple writers can create non-determinism. This can not be resolved in cases where assumptions are contradictory. The advantage of this simulation (execution) based approach is, that this can be detected automatically.

The above explained second mode of assumptions generally can be used for any kind of well-directed automatic test vector generation, in order to prevent false negatives while evaluating the observed test runs.

Environmental C-Drivers or Assumption Observers can be synthesized from any high level assumption specification in the same manner as C-Observers are synthesized from any formal commitment specification.

6 Test Execution and Test Evaluation via Offline Observer-Simulation

The straight forward method of evaluating any tests with the described observer technology simply embeds the C-Observers in the test harness (main program) of the execution platform. In other words, the target-executable is including all C-Observers and the needed result recording functionality. This approach generally has several disadvantages:

- not all target platforms can be used for this approach as the execution speed and the memory size is limited
- the integration effort of the observers in the test harnesses is very high
- even worse, the observers can not be integrated on all targets, e.g. in HiL or real vehicle environments

In order to overcome those issues, yet another method, the so called Off-line Observer Simulation is used. For this purpose, existing test stimuli vectors are executed on the target-execution level (MiL, SiL, PiL, HiL[15] or even in the vehicle) as it is done in the conventional testing approach. Then the reaction of the SUT is recorded on the selected platform in correspondence to the stimuli input vectors of the used test scenarios. The observers are not executed directly on the execution-platform as it could influence the systems reactions as discussed above. Afterwards the recorded test

[15] Hardware in the Loop (HiL) is a technique which allows connecting an embedded system under test (Hardware and Software) to a simulation of the real environment of the system in order to be tested under real conditions. The simulation of the real environment in general is done by very complex and fast Hardware (HiL-Simulator) in order to guarantee real-time aspects during test activities.

vectors (inputs, outputs and observables[16]) will be replayed on a virtual execution platform. In other word, the real evaluation of the performed tests is done off-line. This principle is visualized with the following figure.

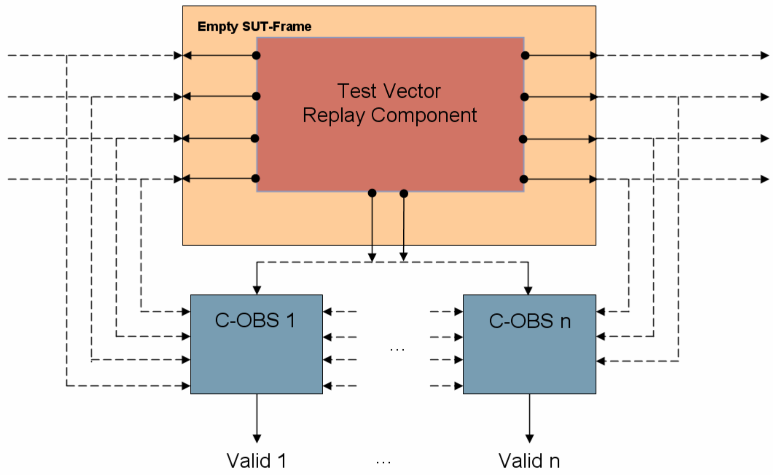


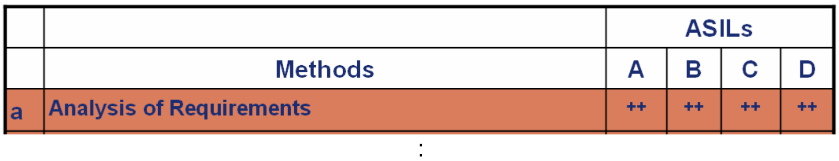
Figure 8: Offline Observer-Simulation

It shows that the VVP is changed. The behavior part of the SUT is replaced by a Test Vector Replay Component. It plays back the recorded interface behavior of the SUT which has been stored on the target execution platform. During this play-back process, the C-Observers are running in parallel with the replay component within the VVP. This is done in order to check if the SUTs behavior is acceptable regarding the specified requirements indirectly through the C-observers.

In some cases, this method is limited by the level of observability of the target platform and the used signal recorder. In most cases, this disadvantage can be prevented by a well designed diagnostic interface of the application under development.

7 Automatic Requirements Based Test Generation

The ISO 26262 recommends (next figure) the analysis of requirements in order to find / derive appropriate tests for the SUT. As described in a chapter above, the requirements are represented by the introduced C-Code Observers. Therefore, C-Code-Observers are used as basis for deriving the appropriate test cases as recommended in the ISO 26262.



		ASILs			
	Methods	A	B	C	D
a	Analysis of Requirements	++	++	++	++

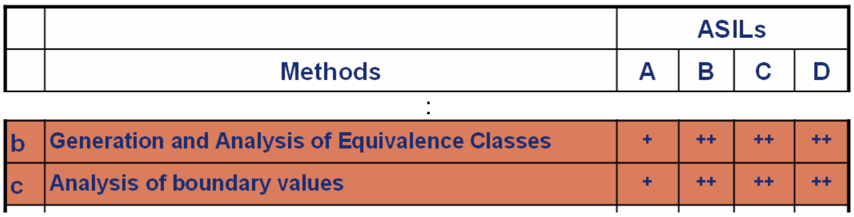
Figure 9: ISO 26262 recommendations regarding Deriving Test Cases Part 1

[16] Observables are variables which can be read in the test harness of the SUT. According to whether the code generator has been forced by the user to make certain signals visible (display variables) even local variables can be recorded on the execution platform.

The VVP semantically based on C-Code consists of all needed components (SUT, Environments and Requirements via C-Observers) to virtually represent the complete systems behavior. This VVP is used as input for the existing test vector generation of the testing tool environment in order to structurally cover the C-Code the observers. This guarantees the generation of requirements based test cases as the complete VVP is taken into account.

8 User Defined Test Cases via C-Observers

Beside requirements based test cases, other methods of deriving important tests are recommended by the ISO 26262. It can be seen in the following figure that the further analysis of Equivalence Classes and Boundary Values are of high interest from ASIL B to ASIL D.



		ASILs			
	Methods	A	B	C	D
	:				
b	Generation and Analysis of Equivalence Classes	+	++	++	++
c	Analysis of boundary values	+	++	++	++

Figure 10: ISO 26262 recommendations regarding Deriving Test Cases Part 2

Any test cases either structural or data driven can be defined by simple branch-points of C-Observers. This means any equivalence class definition and any data boundary value definition can be represented as if-then-else-cascade within a C-Code-observer. In contrast to C-Observers for requirements, these observers are not directly used as a watch-dog which indicates desired or undesired behavior. It is used to allow the automatic test vector generator to cover all important branches of this if-then-else-cascade to fully satisfy the ISO 26262 recommendations regarding test case derivation. After the test case generation process, the generated set of stimuli vectors are used to run a simulation on the reference execution level (e.g. MiL which represents the well tested “Golden Device”[17]) to get the test vectors which includes all reference test data (inputs and recorded observables). In the final process step, these reference vectors are used for a back-to-back test against the SUT (e.g. the final implementation on the processor PiL).

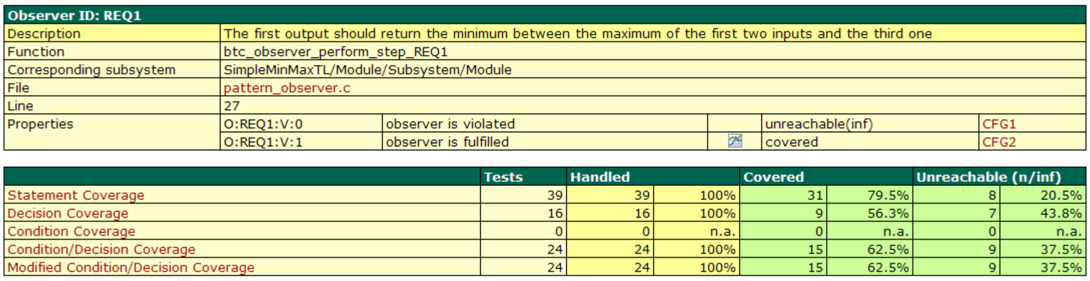
9 Automatic Requirements Coverage Measurement

As the ISO 26262 is recommending a test quality measurement over model coverage, code coverage and requirements coverage, an ISO 26262 compliant mechanism has to be established within the testing tool environment via the VVP. Model- and Code-Coverage measurement is an existing capability of the back-to-back testing use case supported by the testing tool environment. With the extension of C-Observer technology

[17] A “Golden Device” is an ideal example of a device (such as a unit of measure) against which all later devices are tested and judged. The term "golden" is used to describe the precision of the device to standard specifications.

for the new use case automatic requirements based testing it is possible to introduce a new method to measure the missing Requirement Coverage in an automatic way.

All aspects of the requirements are represented within the c-observer definitions. Therefore a complete structural coverage of those c-code-observers is measuring the desired requirements coverage much more accurate than the intuitive approach described in the ISO 26262. In the testing tool environment for each defined C-Observer, the coverage rate is measured at any time of usage. An example can be seen in the next figure.



Observer ID: REQ1					
Description	The first output should return the minimum between the maximum of the first two inputs and the third one				
Function	btc_observer_perform_step_REQ1				
Corresponding subsystem	SimpleMinMaxTL/Module/Subsystem/Module				
File	pattern_observer.c				
Line	27				
Properties	O:REQ1:V:0	observer is violated		unreachable(inf)	CFG1
	O:REQ1:V:1	observer is fulfilled		covered	CFG2

	Tests	Handled		Covered		Unreachable (n/inf)	
Statement Coverage	39	39	100%	31	79.5%	8	20.5%
Decision Coverage	16	16	100%	9	56.3%	7	43.8%
Condition Coverage	0	0	n.a.	0	n.a.	0	n.a.
Condition/Decision Coverage	24	24	100%	15	62.5%	9	37.5%
Modified Condition/Decision Coverage	24	24	100%	15	62.5%	9	37.5%

Figure 11: Requirement Coverage measured via C-Code-Observer Code-Coverage

Beside pure coverage, the term “Handling Rate” has been introduced to define the test-end criteria for requirements based testing. If a specified requirement is valid under all circumstances, the violation branches of an observer can never be covered by any test stimuli vector. This potential violation branches can be seen in a graphical visualization of an observer in the next figure.

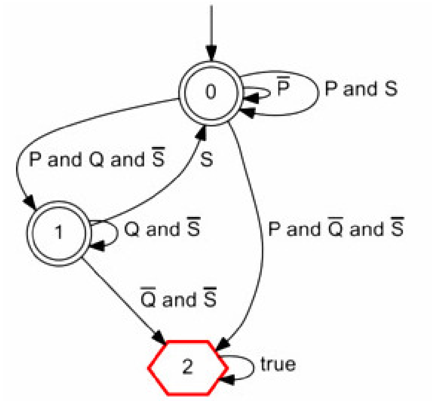

0
$\overline{P}$
P and S
P and Q and $\overline{S}$
S
1
Q and $\overline{S}$
P and $\overline{Q}$ and $\overline{S}$
$\overline{Q}$ and $\overline{S}$
2
true

Figure 12: Visualization of a C-Observer[18] with two possible branches representing a failure situation

Therefore the structural coverage rate can never reach a hundred percent. The testing tool environment with its verification engines can also prove[19] the unattainability of certain branches. Hence, that these dead-braches can be “handled” means they are indeed completely analyzed by the verification technology.

[18] An observer consists of Accepting States and a single Failure State to represent wrong system behaviour during observation. The transitions to the red failure state may be never taken, if the corresponding requirement is fulfilled by the SUT.

[19] This capability is available for a subset class of C-Code of the SUT. The C-Code currently has to be integer or fixed-point code, in order to be able to finally prove the absence of specific branches. In the future, maybe other technologies will allow extending this capability even up to floating point code.

10 Conclusion

The presented extension of the existing test and verification tool environment to support the automatic requirement-based testing in an ISO 26262 compliant way is another huge step forward in solving the testing problems of nowadays within the existing MBD processes in automotive.

It addresses the need for a maximum of test automation while improving the target quality while directly fulfilling the relevant industry standards concerning functional safety.

First experiences in pilot projects show that this seamless test and verification approach for requirements based testing not only improves the target application quality and saves a tremendous amount of human test effort, it additionally supports and improves the OEM/Supplier relationship processes. As the presented observer technology can be used along the whole V-Process, from early stages of the requirements capturing phase to the final product implementation, it improves the communication chain between the different development and test stages. This is especially the case within existing well established MBD Processes, which is accompanied by a central data base including and manageing all work products of the development process.

Future extensions of the presented observer technology go in the direction of an automatic on-board diagnostic approach, where the C-Code-Observers are reused for the implementation of the diagnostic components of a vehicle. So the observers finally could be part of the implementation of the embedded system itself. This allows full traceability between the field experiences of the final product and the other development work products like requirements specifications and functional model etc. This will improve the product even over the lifecycle boundaries.