Test requirements in networked systems

Jürgen Klüser, Vector Informatik GmbH

The use of CAN with J1939 or CANopen based higher layers leads to cost efficient and flexible solutions, but together with a high increase of the electronics' complexity. An additional complication is the typical approach of distributed development between OEMs and several suppliers. The consequence has to be a systematic improvement of the development process. Project risks are to be reduced by taking testability as a design requirement and by performing the appropriate tests in the very early project phases. This paper discusses concepts combining system prototyping with test case generation along the V-model.

1 Introduction

The development process of a networked system can be described by the V-model. Figure 1 shows the typical situation of today's testing activities in a simplified V-model.

In contrast to pure theory the reality shows a delay of the implementation phase. Testing activities, even if planned in the beginning, are going to be specified just before they have to be performed. A typical argument is that the implementation phase will bring expertise and even changes in the original design and maybe even a more detailed view on the requirements. The consequence of late test specification is a high risk. Due to the reduced time remaining, this most often will delay the overall project – or testing will not be performed sufficiently. Problems in the implementation or in the design, found at this late stage cannot be solved in time and tested again.

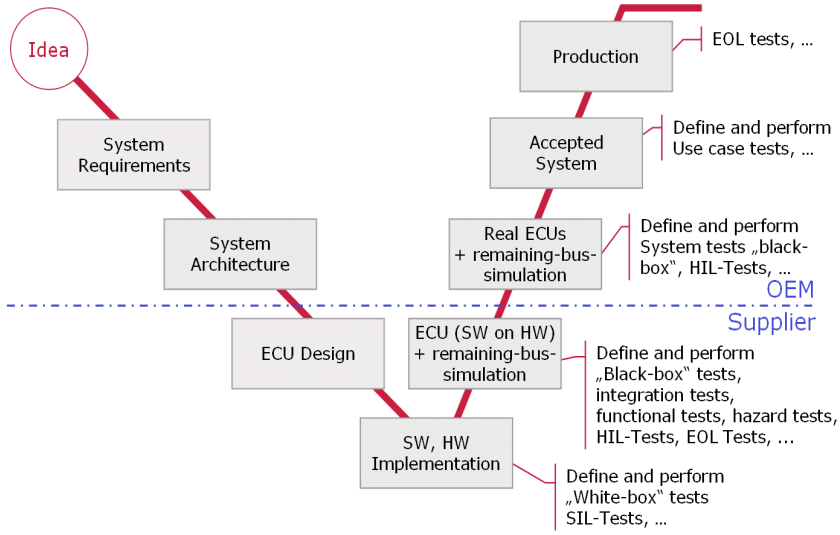


Figure 1: Today's testing activities in the product's life cycle

2 Testability as a system requirement

The V-model includes two basic patterns. The first says that every phase is identified by results that have to pass a quality gate. The second demands that the phases and their quality gates on the right side are a direct traceable consequence of the corresponding phase on the left.

The System Requirements mainly define "use cases". This clearly should include topics like testability as an input for the architecture, requirements for EOL tests, and diagnostics requirements. Applying the two basic patterns to that approach of testability as requirement forces the system architect and designer to provide measures and specifications for tests in the early phases.

Experience with that approach in the diagnostic domain as presented by [3] showed a significant increase in quality and process stability. This can be applied to the testing domain in the very same manner.

3 System Architecture – Introducing an executable specification

For years it is state of the art to describe the system architecture with simulation tools as a simulated model. Besides manual review technology this allows to systematically verify many aspects of the design. Consequent application of this approach leads us to the concept of the executable specification. This is true also for the testing domain (Refer to [4]).

Specifying tests in an executable form in fact leads to a first implementation level of these tests. This immediately brings a set of benefits:

The system architecture model can be tested systematically. The real world shows that problems detected in later phases may require slight modifications in the design. The tests can then be applied as regression tests. In formal words this means passing the Quality Gate Architecture can be achieved with less effort on a higher maturity level.

Even the test concept itself can be tested in an early phase. A valid argument against writing test specifications before the verification and validation phases had been that the implementation will almost always bring modifications and changes. These changes invalidated test specifications and therefore resulted in wasted effort. An executable test spec can be changed without the need to specify everything from scratch again.

From the communication's perspective the executable specification has further advantages: After definition of the relations between ECUs a big part of the communication description can simply be generated. This is even more true in those cases where a higher layer protocol such as J1939 or CANopen is used. These standards give much information about how relations have to be realized. In J1939 for example the signals are assigned to concrete parameter groups, that in most cases have well-defined communication parameters. The object model of CANopen allows one to generate the complete communication behavior after selection of the object relations. But tools with build-in knowledge of these protocols do not only generate complete communication system simulations – in addition they generate many test scenarios, that are ready-to-use. From pure "use-case tests" as the highest layer this can go downwards to "protocol tests" and "communication tests".

The usage of typical commercial protocol stacks also requires testing of the integrated product. The flexibility for being usable in many different applications in protocol stacks is implemented by configurability – which may be done incorrectly by the integrator. The protocol tests allow verification of conformance to the higher layer standard.

This generation concept does not only save time and enhance maturity once. After potential changes the test cases can

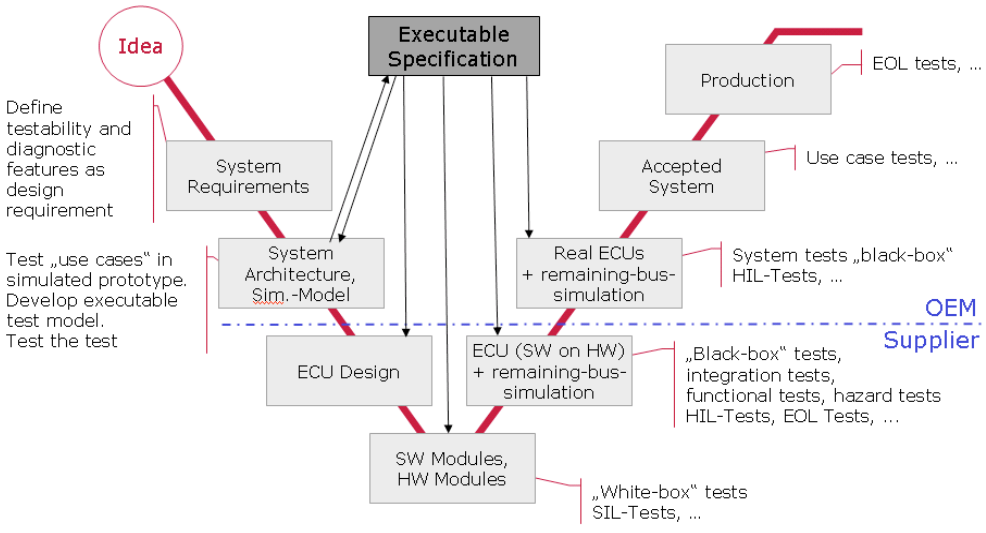


Figure 2: Approach with an executable test specification

be regenerated automatically. This will require minimal manual adaptations – if at all.

Refining the system model by a basic behavior description in fact specifies the requirements as an input for the ECU design. In parallel the test specifications will be refined. These are to be passed to the ECU suppliers (external company or internal departments). Today they are faced with the problem of written specifications, which very often do not describe specifically enough of what is needed or which are not fully consistent. Improved by the concept of passing simulations the full strength develops with having efficient test cases available that are part of the development contract.

Regardless if the behavior description is model driven or conventional, regardless of the abstraction level – in any case the simulation tool must support this by a built-in description language or by an interface to external behavior modeling – or the best of both. Whether this is UML based, XML based, proprietary or whatever goes beyond the scope of this article.

4 ECU development

With a written specification, designing, implementing and testing it on that basis only, is pure theory. In reality the developers need an environment with which they can implement and test stepwise. Like a debugger or emulator for the SW environment the remaining bus simulation for the communication part most often is the only chance to provide a sufficient environment. For example, implementing and testing operation modes of an ECU is not possible without correctly working network management.

A typical problem in these small development steps is the availability of the counterpart of the communication modules. It costs much effort and time to set-up or implement the tests. For many cases standard analysis tools are a good choice. Much more efficient is the support by a dedicated test tool. It should provide the generated tests not only as a set but should allow for the selection, grouping, and execution of single tests easily. To accomplish this each test needs clear pre-conditions and constraints.

Specifically the protocol tests and communication tests help to detect problems very early and therefore reduce the number of iterations between implementation and verification phases.

5 Re-use and refinement for the ECU verification

The final phase of the ECU development is the ECU verification. Potential modifications of the original specifications caused by implementation experience lead only to modifications of the executable prototype. These immediately can be applied in the form of the “remaining bus simulation” concept. And the same is true for the modeled tests.

For many higher layer protocols user organizations provide conformance tests, for example the CANopen conformance test tool of CiA, the DeviceNet conformance test of ODVA, or the ISOBUS test of DLG. It is strictly recommended to use such support. It guarantees that an ECU fulfills a certain level of conformance and interoperability. Anyhow, an intended drawback is that if a test fails, the tools do not really help in finding the reasons. This is a gap that has to be filled by the dedicated development test tools. So, once again the ability to perform single tests with meaningful tracing and problem tracking – and this with fast turnaround times – is indispensable.

As an example take CANopen: If the SDO (transport protocol) implementation has a problem accessing an object, the conformance test just will tell you “SDO failed”. Repeating this in order to find details will require a long turnaround time again. A development tool with a test feature set and its generated “protocol test” will quickly tell, what messages have been wrong and can support in finding the problem source even down to the bit level.

As already demanded for the ECU design, the ability of a test description language is essential here as well. In contrast to the pure conformance test tools the test procedure needs to be extendable by ECU specific or application specific features.

This furthermore makes one independent of the set of tests, the tool provider implementation, and allows for future extensions.

6 Back to the OEM

A strong argument for remaining bus simulation is to make the system integrator capable of starting the integration of an ECU before all other ECUs are available. Features like simulated network management let the ECU think it was in a real environment. This is also a precondition for starting the first integration tests. This will save time for example if delivery delays of single ECUs occur. The automatic tests can be applied. Final integration will then only require regression tests. The focus here will be on the level of the communication tests, while the use-case tests are more a subject of the validation and acceptance phases.

7 Some detail aspects

The behavior of an ECU depends on its whole environment. Essentially known from the HIL concepts (see [1], [2]) any test environment needs to fulfill that aspect. As shown in Figure 3 this requires the ability to treat all relevant bus systems, but also to observe and control digital and analog I/O. This includes the treatment of standard and OEM specific system and device descriptions like DBC (CAN), LDF (LIN), EDS (CANopen), CDD/ODX (Diagnostics) and many more. Besides a GUI it shall be able to define and use user-defined panels and an open interface to application specific software.

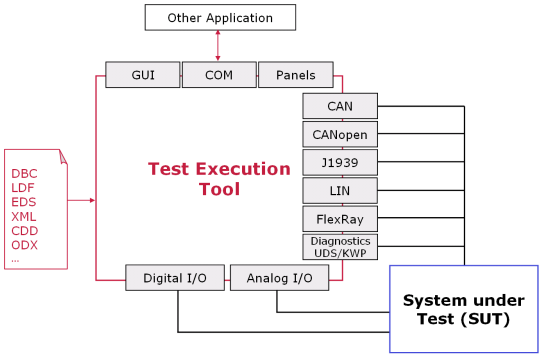


Figure 3: Testing domain

For White Box tests it is desirable to have an interface to debugging tools (Figure 4). For example in car ECUs the RTOS OSEK is used quite often. Here for example debugging and control of task states is essential.

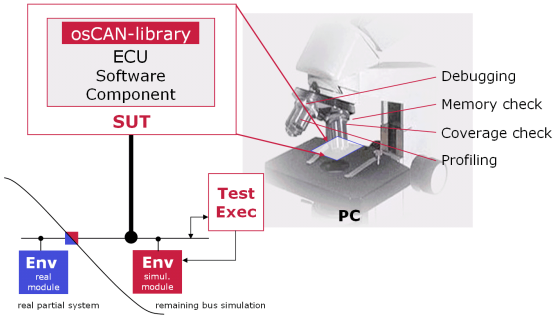
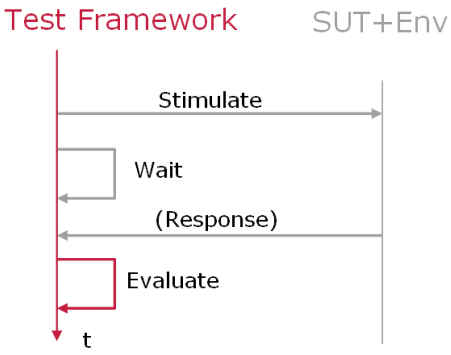


Figure 4: White box test support

As mentioned above, for Black Box tests it is important to have various possibilities to efficiently describe the test activity, sequence, and timing. This should be selectable from scripting (Figure 5) or with a more abstract test pattern approach (Figure 6).




```
// Stimulate
output(Req2MsgBuf)
SetSignal(Req1Msg::SigX, 5)

// Wait
TestWaitForMessage(RespMsg, 2000)

// Evaluate + report
TestValidateSignalInRange(RespMsg::SigA, 0, 1)

GetWaitEventMsgDataCAN(RespMsgBuf)
If(RespMsgBuf.Time > lMaxTime) {
  TestStepFail(„Time“, „Too late“);
}
```

Figure 5: Scripted black box tests

In both cases the support has to include basic bus and I/O operations like output bus messages (CAN frames, MOST frames, J1939 PGs etc.) but also signal operations. The latter can be achieved by a so-called Interaction Layer. This has the task to map signals to the appropriate

communication entities with the correct communication parameters (timing, trigger) according to the system description.

For easy treatment of timing conditions the system has to support event conditions with Wait constructs for messages, environmental conditions, and signal conditions in any combination.

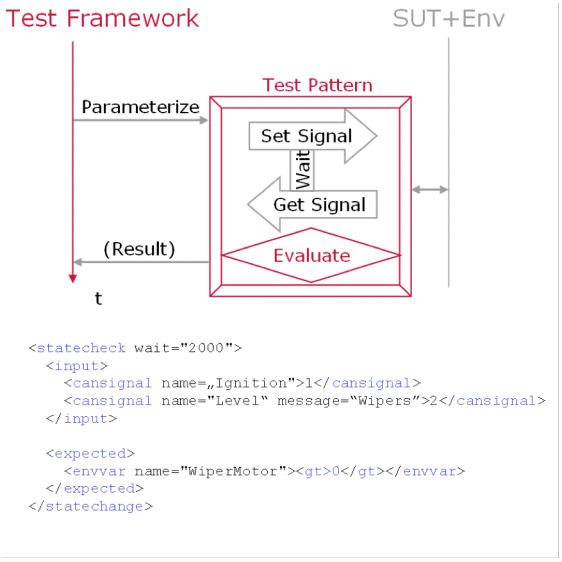


Figure 6: Test pattern approach

Meanwhile XML is widely used . There are many highly sophisticated tools for editing, checking and formatting XML. Figure 7 shows an example for editing a test case with Altova AuthenticDesk. The user does not need knowledge about the XML structure itself or about complex scripting languages.

The open structure for tools can be utilized for test generation tools too. Typically domain know-how exists in various forms. Examples for test-case generations are:

DOORS: Structuring requirements from which test-case can be organized.

UML Tool: Use sequence and/or state diagrams

CANdb++: Monitoring of communication parameters like cycle times, message DLCs, mapping of signals.

CANdela Studio and DiVa: Specify diagnostic services and generate the necessary tests.

DaVinci: Test mapping of all input and output signals to ECUs.

ProCANopen: Generate complete CANopen protocol tests.

… and many more.

8 Reporting

Last but not least – all that test support is useless without a thorough reporting of the results. A modern test tool will provide automatic reporting. A typical test control structure is shown in Figure 8. It substructures in test modules with global invariants. Here again the strength of XML can be used. By using XML for the test patterns and specifications, groups of tests, test cases and sub-test-cases including their respective invariants the test tool will control the test execution. At the same time it will

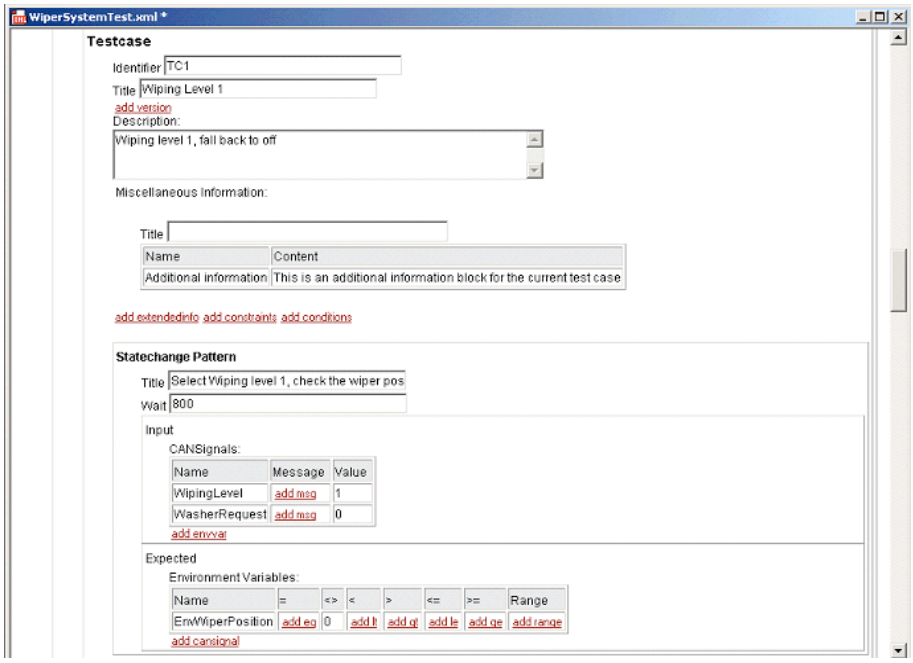


Figure 7: Test description in Altova AuthenticDesk (XMLSpy)

set-up the report structure. The report will be created automatically, formatted by a style sheet. It contains the test structure, the test cases, the overall result, and the test passes and fails with their respective environmental conditions.

9 Conclusion

The impact of consequent test concepts very often is neglected in the early phases of a project. For designing, specifying, implementing, and executing tests near the end of the project, the typical lack of time in that phase is a high risk for the project delivery date, the costs, and in particular for the quality of the product. Applying concepts of model-based development, rapid prototyping, and code generation leads to an executable test specification.

This shifts much of the work to earlier project phases, provides a testing environment already before the implementation and at the same time avoiding double-work for test specification changes due to design changes.

The quality of all phases is significantly enhanced, the efficiency of the implementation phases is increased, and the project risk and cost is reduced.

References

[1] Siegfried Beeh, Vector Informatik: Testing with CANoe, Vector Congress 2004

[2] Thomas Bardelang, DaimlerChrysler: CAN in the HIL for the new Actros; Experiences in the test of networking mechatronic systems, Vector Symposium 2004

[3] Norbert Schlingmann, Claas: Challenges and methods in the development process of CAN based diagnostic systems, Symposium CAN in Commercial Vehicles 2005

[4] Mirko Tischer, Vector Informatik: Prototyping and testing CANopen systems, CAN Newsletter 3/2006

[5] Vector Informatik GmbH, CANoe user manual, 2006

Several Figures are taken from [1]

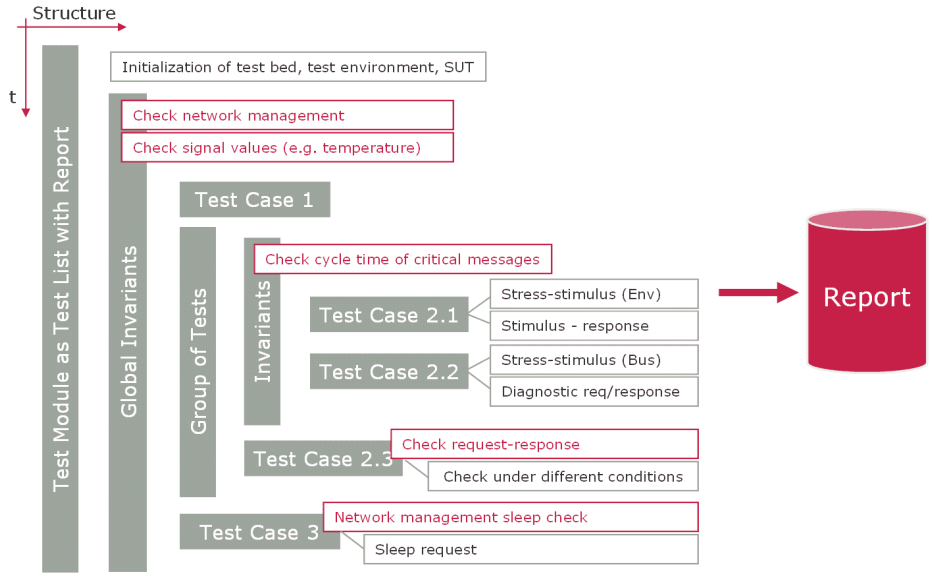


Figure 8: Test control and report structure