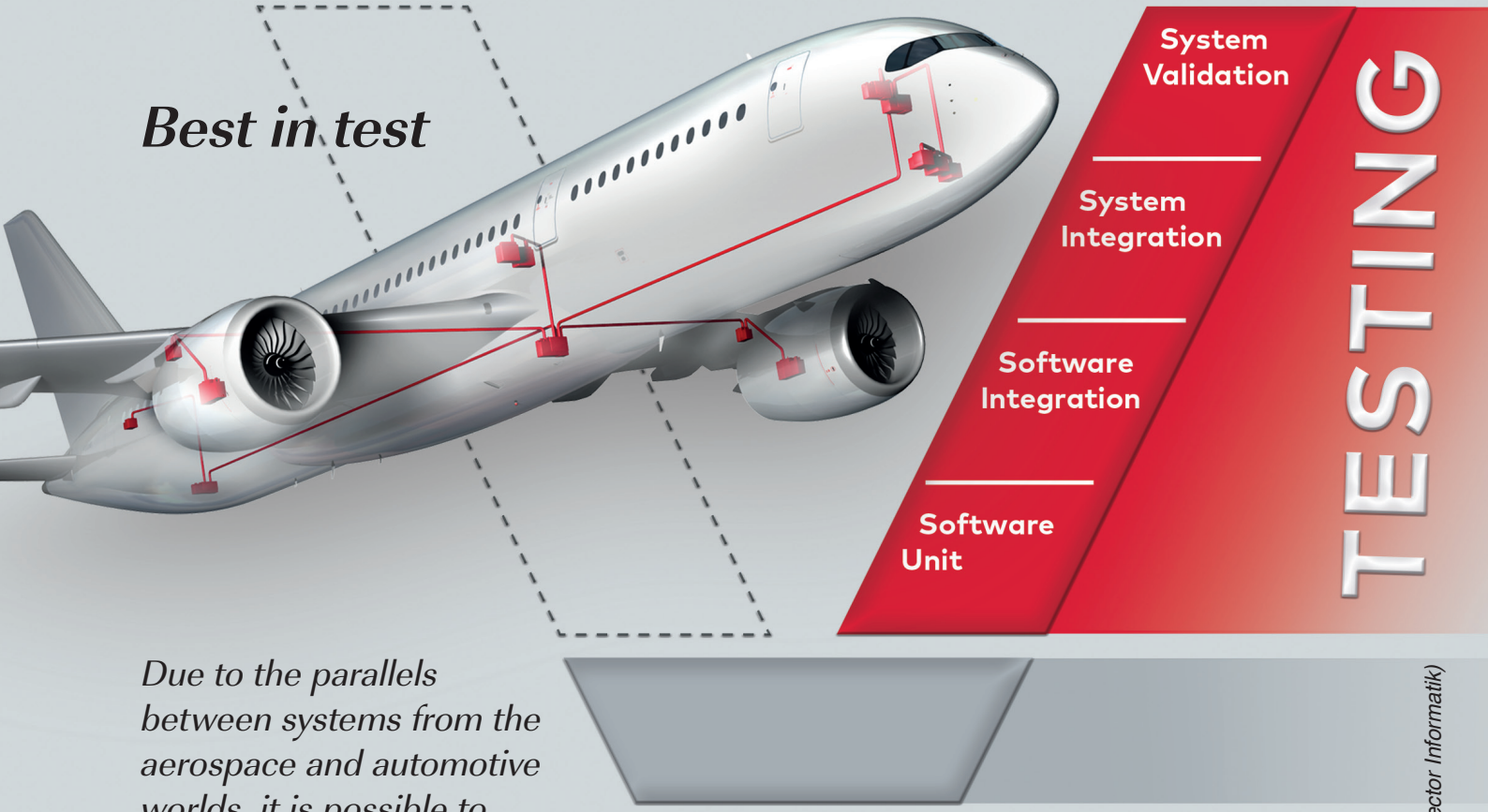
Best in test



Due to the parallels between systems from the aerospace and automotive worlds, it is possible to transfer proven concepts and processes from the automotive industry to avionics. Vector describes it's approaches and concepts.

(Source: Vector Informatik)

Continued development of comprehensive and structured testing methods and tools for electronically networked aircraft and cabin systems is not only necessary for economic reasons. With the recent safety-critical pilot assistance systems such as Enhanced Vision and Runway Overrun Protection Systems or with "wireless" cabin functions each requires appropriate testing strategies that are compliant to the regulatory rigors assigned to them (i.e. DO-178C). This article describes Vector's approaches and concepts.

The software in avionics and ground-based systems are bounded by strong regulatory standards DO-178C and DO-278 respectively. With failure regarded as "not an option", significant analysis and effort is put into the verification and validation of these systems. In fact, industry wide, in a typical project fifty percent of the development budget is used for structural testing the software according to Federal Aviation Administration (FAA) DO-178C Level A [1]. The ability to automate and simulate these systems can greatly assist in reducing the overall effort, and hence the implementation costs.

There are three major phases of verification and validation in avionics and ground-based software (Figure 1): unit testing, integration testing, and system/functional testing. In each phase, test cases need to be derived from their appropriate level of requirements with full traceability between both.

While the concepts and methodologies for low level testing have been reasonably consistent over the years, the introduction of more networked systems based on the CAN and AFDX protocol, and the drive for code reuse, demands innovations in the approach as to how the software should be tested. To find good solutions, other industries can be considered that have successfully deployed complex networked systems, with rapid time to market demands and highly critical functionality. An example is the automotive market, with its drive by wire systems, autonomous vehicle technology, 18 to 24 month development cycle and CAN/Ethernet networked platforms.

The similarities in particular in CAN-based systems make it possible to transfer proven concepts and processes from the automotive industry into the avionics domain. CAN is currently used in modern civil aircrafts like A350 and Boeing 787 for systems such as environmental control, doors, galleys, smoke detection, potable water, and de-icing. Furthermore young companies acting in the emerging market of hybrid and full electric air vehicles for new urban air mobility concepts rely on CAN-networks as well.

Due to the specific challenges like long cables, extreme environmental conditions, stringent lightning protection requirements, and long service life, adequate test strategies at all test levels must be foreseen.

The approaches can be considered at three levels as described in section 6.4.3 of the DO-178C standard: low level testing, software integration testing, and hardware/software integration testing. Finally, it is worth considering how these can be coupled into a process which provides greater agility as well as introducing shift-left strategies into the development process. ▷

Low-level testing

This testing level is used to test the low-level requirements and is usually accomplished with a series of unit tests that allow the isolation of a single unit of source code. To test a single unit in isolation, a huge amount of framework code such as test drivers and stubs for dependencies (Figure 2) must be generated. Ideally, this should be done automatically with a tool that offers an intuitive and simple approach for defining test scenarios. This meets the main requirements of section 6.4.2 “requirements-based test selection” and the sub-sections “normal range test cases” and “robustness test cases” of the DO-178C standard. With the growing need for code reuse, it is very likely the same unit of source code might be used in several configurations. Therefore, it is important that the definition of a test case is not tightly coupled to the code and provides flexibility in how they can be maintained as the software evolves over time. Typically, the use of a data driven interface for the definition of test cases has proven to be more maintainable over time than a source code definition.

This approach also means that when the source code and associated test cases are deployed in a continuous delivery workflow, as changes are made to the code, the testing framework can quickly be regenerated and the test cases appropriately remapped. Where significant changes have been made, these can be flagged for further review without breaking the rest of the automated workflow.

A good example of this is the embedded software testing platform Vectorcast, that automates testing activities across the software development lifecycle. It fully supports testing on target or using the target simulator normally provided by the compiler vendor. Structural coverage from testing isolated components can be combined with the coverage gathered during full integration testing to present an aggregated view of coverage metrics.

Vectorcast test cases are maintained independent of the source code for a data-driven test approach. This technique allows tests to be run on host, simulator, or directly on the embedded target in a completely automated fashion.

Software integration testing

Software integration testing verifies the interrelationship of components. This concept is also known as software-in-the-loop (SIL) testing. The idea here is to bring the software components together and test them without any of the complexities of the underlying hardware. A critical aspect of testing software during this phase is the ability to simulate dependencies and interfaces in the integrated unit that is under test.

To simulate this software conveniently, it is common to use a host-based compiler like Visual Studio, GCC, MinGW, etc. to run the code, and then once a level of confidence has been achieved, the cross-compiler can then also be used. Depending on the certification level for DO-178C in Level C, B or A, certification credit for the activity may only be permissible when done using the cross-compiler and running on the target.

In the low level testing framework, the collection of software units can still only be tested via programming ▷

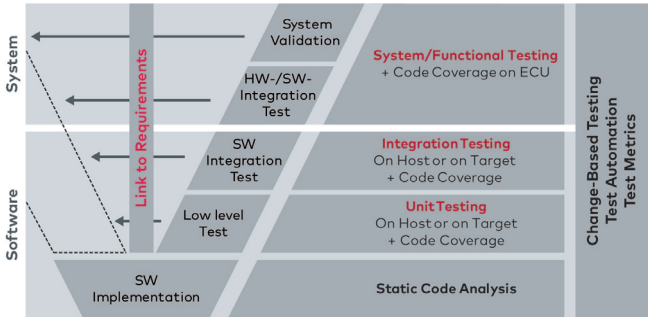


Figure 1: The major phases of verification and validation in avionics and ground-based software (Source: Vector Informatik)

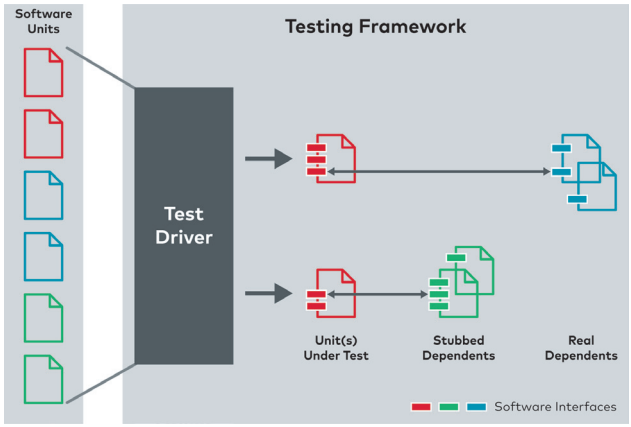


Figure 2: Low level testing framework to test a single software unit in isolation – using framework code such as test drivers and stubs for dependencies (Source: Vector Informatik)

API calls. In this case the use of a test automation framework like Vectorcast is ideal, as it will automatically build the required drivers and stub any units that are outside the units of interest automatically. There could also be an opportunity to reuse some of the test cases from low level testing for units that are higher in the call tree.

Alternatively, the software components to be tested may be closely reliant on the underlying hardware, and a more robust simulation of the underlying hardware is required to correctly verify the software functionality in this case.

Hardware/software integration testing

This type of testing is used to satisfy high level requirements and is performed on the target hardware using the complete executable image. The challenge when testing at this level is to provide enough external stimulation to the line replaceable unit (LRU) such that it functions correctly. The external simulation comes in various forms: logical pins, avionics data network, modeling tools, etc. Additionally, because of the complex nature of the networks, it should also be possible to easily extend or customize the simulation interfaces quickly and easily.

An example system to validate an LRU at this level can be setup using the tools VT System and CANoe (Figure 3). The software and hardware combination CANoe and VT System from Vector offers a test system that can be scaled from simple test equipment at the developer workstation to the highly automated HiL environment in the test lab. The core idea of the VT System of is to combine all the hardware functions required for LRU testing in a modular system seamlessly integrated into CANoe. The test hardware covers the inputs and outputs, including the power supply and network connections of a control unit or subsystem. At each pin, the pin function according to stimulation, measurement, load simulation, fault connection, and switching between simulation and original sensors and actuators are possible. These functions are so universally designed that a once constructed test system can be used for different LRUs.

In CANoe, in addition to the network environment, the physical environment can also be simulated using appropriate Matlab / Simulink models. A closed hardware-in-the-loop simulation is just as possible as a simple, manual stimulation without elaborate models. CANoe offers the same flexibility in test automation. The tool Vteststudio provides a modern authoring tool. The possibilities to define tests range from programming in various languages like the Vector own Capl and .NET/C# over defining simple test procedures in tabular form to graphically noted test models. It is used to define test procedures and allows the developer to flexibly combine the different input methods.

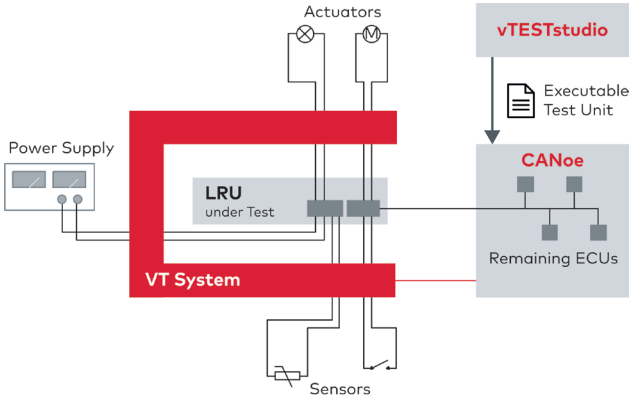


Figure 3: Example setup for simulating the environment around an LRU (Source: Vector Informatik)

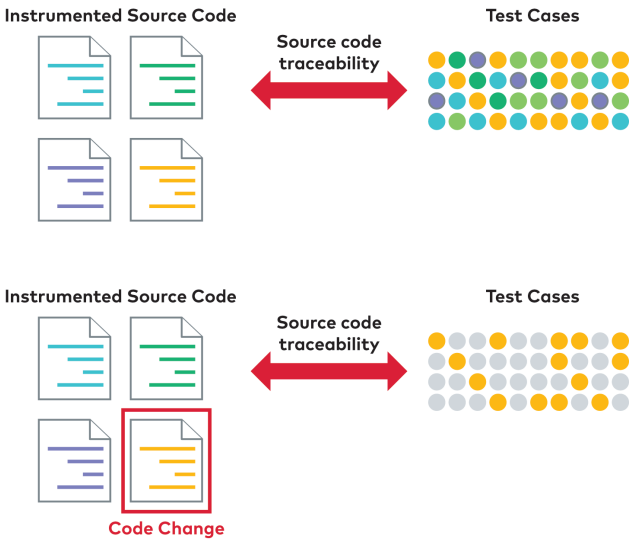


Figure 4: Change-based testing greatly reduces testing time while ensuring testing completeness (Source: Vector Informatik)

The finished test sequences are stored as test units and are then executed in CANoe.

CANoe executes the test cases and at the end of each test run the system creates a detailed test report. Finally, all threads from test and execution planning to execution documentation converge in test data management. This always ensures good traceability.

Bring it all together to enable a lean continous integration platform

One of the biggest challenges with software development today is the unintended propagation of defects or issues through the development cycle of a system. These issues can often be identified very early in the development cycle but are missed because the software is merged without the adequate verification and validation in place. To address this quality issue, there are various discussions on the topic of 'shift left', i.e. test earlier in the process. However, in general the time required to rerun all low level, software integration, and hardware/software integration tests can be very time consuming. In some cases, a complete end to end run of all test cases can take between three weeks to as long as two months. This time frame does not fit the rapid feedback that is required to provide developers with early feedback of issues that they might have introduced at the time of writing the software.

To address this challenge, the concept of change-based testing (CBT) can be introduced. This method helps organizations test faster and smarter by analyzing each code change against all existing test cases and choosing the sub-set of tests that are affected by the change (Figure 4). By running only this sub-set of tests, test execution times are greatly reduced, and developers get immediate feedback on the impact of their changes. This allows bugs to be fixed immediately, when they are introduced, rather than weeks later, during "full" testing.

By using a test automation platform like Vectorcast, structural code coverage is collected from all levels of testing like low level, software integration, and hardware/software Integration. The ability to integrate the code coverage reporting with software integration and hardware/software integration tools like CANoe and VT System provide a single perspective of the system's aggregated code coverage, and how a specific test directly contributed to the overall code coverage. Thus, when a change is made to the underlying software, the Vectorcast decision engine quickly computes the impacted tests at all levels and dispatches them appropriately – even when the test is to be run through CANoe or VT System. Running a subset of tests represents a significant time saving in the execution time, and shortens the time taken to determine an impact from a change made to a matter of hours with a high level of confidence.

Conclusion

As the complexities of avionics and ground-based systems continue to evolve, the need to provide more sophisticated strategies and tooling for addressing the compliance required for verification and validation for DO-178C and DO-278 will continue to grow. The networked aircraft will require the ability to not only ensure that a single LRU functions correctly, but all LRUs also function correctly when the entire system is brought together. This means that the ability to isolate components at a software unit level, as well as a LRU level while simulating the remaining interfaces will be critical to achieving the quality requirements of the avionics industry. Furthermore, the artifacts from the verification and validation activity can be integrated into a continuous integration process to introduce modern 'shift-left' concepts into the development of safety critical systems while ensuring compliance to the standards. ◀

References

[1] https://www.vectorcast.com/customers/testimonials/lockheed-martin-uses-vector-softwares-vectorcast-ada-c130j-do-178b-testing

Authors

Dr. Arne Brehmer, Hans Quecke
Vector Informatik
info@vector.com
www.vector.com