Low-level testing is used to test low-level requirements and is usually accomplished with a series of unit tests that allow the isolation of a single unit of source code. While the concepts and methodologies for this type of testing have been reasonably consistent over the years, the introduction of more networked systems based on the AFDX (Async Time-Division Switched Ethernet) protocol, and the drive for code reuse, demand innovations in the approach to testing software. Finding good solutions means looking at other industries that have successfully deployed complex networked systems, with rapid time to market demands and highly critical functionality. One such example is the automotive market, with its drive-by-wire systems, autonomous vehicle technology, 18- to 24-month development cycle and CAN/Ethernet networked platform.

The similarities in these systems make it possible to transfer proven concepts and processes from the automotive industry into the avionics domain. The approaches can be considered at three levels, as described in Section 6.4.3 of DO-178C: low-level testing, software integration testing and hardware/software integration testing. Finally, it is worth considering how these can be coupled into a process that provides greater agility and introduces shift-left strategies into the development process, which broadly means to test earlier.

LOW-LEVEL TESTING

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 requirements of Section 6.4.4, Requirement-based Test Selection, and the sub-process Normal Range Test Cases of DO-178C. With the growing need for code reuse, it is very likely that a certain section of source code will 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 the code can be maintained as the software evolves. Typically the use of a data-driven interface for the definition of test cases is 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, they can be flagged for further review without breaking the workflow.

A good example of this is the Unit Test Automation tool VectorCAST. The tool fully supports testing on targets 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 independently 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

This concept verifies the interrelationship of components and is also known as software-in-the-loop (SIL) testing. The aim 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 under test.

To simulate this software conveniently, it is common to use a host-based compiler like Visual Studio, GCC or MinGW to run the code, and open a level of confidence that has been achieved the cross-compliler can be used. Depending on the certification level for DO-178C (Level A, B or C), certification credits for the activity may only be used when done using the cross-compiler and running on the target. In the low-level testing framework, the software unit can still only be tested via programming API calls. In the case a test automation framework like VectorCAST is ideal, it will automatically build the required drivers and stub any units that are outside the units of interest. 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 hardware is required to correctly verify the software functionality.

HARDWARE/SOFTWARE INTEGRATION TESTING

This is performed on the target hardware using the compiled executable image to satisfy high-level requirements. The challenge when testing at this level is to
software testing

Low-level testing

Software testing

AEROSPACETESTINGINTERNATIONAL.COM // SHOWCASE 2019

An example system to validate an LRU is shown in Figure 3. The software and hardware combination offers a test system that can be scaled from simple test equipment to a highly automated hardware-in-the-loop (HIL) environment. The core idea of the VT System 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 subassembly. At each pin, changing the function according to simulation, measurement, load simulation, fault connection, and switching between simulation and original sensors and actuators is possible. These functions are universally designed to such a degree that, once constructed, the 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 HIL simulation is just as possible as a simple, manual simulation without elaborate models. CANoe offers the same flexibility in test automation, while tESTSimics provides a modern authoring tool. Tests can be defined in different programming languages like Vector’s own CAPL, and MATLAB. Furthermore, test procedures can be described in tabular form or graphically using test models. It is used to define test procedures and enables the developer to flexibly combine the different input methods. The finished test sequences are stored as test units and 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 ensures the traceability of test-related issues.

TACKLING TEST COMPLEXITY USING VECTOR TOOLS IN ALL TEST PHASES

As the complexities of avionics and ground-based systems continue to evolve, the need to provide more sophisticated approaches and tools to address test and verification requirements continues to grow. The networked aircraft will require the ability not only to ensure that a single LRU functions correctly, but also that they all 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 at an LRU level while simulating the remaining interfaces, will be critical to achieving the quality requirements of the avionics industry. Furthermore, the artifacts from verification and validation can be introduced into a continuous integration process to introduce modern shift-left concepts into the development of safety-critical systems while ensuring compliance with the standards.

Dr. Arne Brehmer is head of the corporate business area, Hans Quieke is head of the aerospace business area, and Hans Wibmer is product manager for networking and communication design and production at Vector Informatik.