Complete Verification and Validation for DO-178C

Whitepaper | V1.0 2019-10

Table of Contents

1 Purpose

This whitepaper describes how the Vector software testing platform is used to satisfy the Software verification process objectives as defined in section 6.0 of the DO-178C standard, “Software Considerations in Airborne Systems and Equipment Certification.” This whitepaper also discusses Section 12, the software tool qualification process.

2 What is the DO-178C Standard

DO-178C replaced DO-178B to be the primary document by which the certification authorities will approve all commercial software-based aerospace systems. It represents a revision to DO-178B considering the experiences and information gathered for developing software for avionics. The new document, entitled DO-178C (ED-12C), was completed in November 2011 and approved by the RTCA in December 2011. The new document became available for use in January 2012.

The DO-178B was first published in December 1992 by RTCA, Incorporated. The document outlines the guidelines used by organizations developing airborne equipment and certification authorities, such as FAA, EASA, and Transport Canada. The development of DO-178B was a joint effort of RTCA and EUROCAE who published the document as ED-12B. Several certification authorities software team (CAST) papers were developed as clarification papers after the initial publication.

DO-178C prescribes a process to be followed in the development of airborne systems. One of the key requirements in the software verification process of DO-178C is achieving structural code coverage in conjunction with the testing of the high-level and low-level software requirements.

Based on a system safety assessment, failure condition categories are established. These failure condition categories determine the level of software integrity necessary for safe avionics operation. DO-178C classifies software into five levels of criticality based on whether atypical software behavior could cause or contribute to the failure of a system function. The table below shows the relationship between the failure condition category and the structural coverage objective as defined by DO-178C.

Level	Failure Definition	Associate Structural Coverage
A	Software resulting in a catastrophic failure condition for the system	Modified Condition/Decision Coverage, Decision Coverage & Statement Coverage
B	Software resulting in a hazardous or severe-major failure condition for the system	Decision Coverage & Statement Coverage
C	Software resulting in a major failure condition for the system	Statement Coverage
D	Software resulting in a minor failure condition for the system	None Required
E	Software resulting in no effect on the system	None Required

Table 1: Design Assurance Level

3 Testing Approaches for DO-178C

The software verification process objectives are defined in section 6.0 of the DO-178C standard. The approach for testing can be considered at three levels as described in Section 6.4 of the DO-178C standard: Low-level testing, software integration testing, and hardware/software integration testing.

Low-level Testing

Low-level testing 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. VectorCAST is used during this testing phase.

Software Integration Testing

Software integration testing verifies the interrelationship of components. Testing at this level is performed with VectorCAST to test multiple software components under test at one time. The complete test harness is automatically generated to support this kind of testing and software requirements can be tagged to specific test cases to ensure that all requirements are being tested.

Often the behavior of the software under test is reliant of other parts of the distributed system. CANoe can then be used to simulate the behavior of these remaining parts.

Hardware/Software Integration Testing

This type of testing would be used to satisfy high-level requirements and is performed on the target hardware using the complete executable image representing a Line Replaceable Unit (LRU). LRU's typically require external stimulation and simulation to correctly function. The external stimulation comes in various forms: logical pins, avionics data network, modeling tools, etc. Tools like VT System and CANoe can be used to provide this external stimulation and simulation, while VectorCAST/QA can be used during this type of testing to capture the code coverage during execution of system or functional level test procedures.

4 Addressing the Requirements for Software Testing

Section 6 discusses the software verification process for DO-178C. The area specifically of interest in subsections 6.4 relating to software testing, 6.5 relating to traceability and 6.6 relating to configuration parameters.

> 6.4 Software Testing
>> 6.4.1 Test Environment
>> 6.4.2 Requirements based Test Cases
>>> 6.4.2.1 Normal Range Test Cases
>>> 6.4.2.2 Robustness Test Cases
>> 6.4.3 Requirements based Testing Methods
>> 6.4.4 Test Coverage Analysis
>>> 6.4.4.1 Requirements based Test Coverage Analysis
>>> 6.4.4.2 Structural Coverage Analysis
>>> 6.4.4.3 Structural Coverage Analysis Resolution
>> 6.4.5 Reviews and Analyzes of Test Cases, Procedures, and Results
> 6.5 Software Verification Process Traceability
> 6.6 Verification of Parameter Data Items

Each sub-section will be addressed from the low-level testing, software integration testing and hardware/software testing perspectives described previously in this document.

4.1 Section 6.4.1 – Test Environment

This section specifies that more than one test environment may be needed to satisfy the objectives for software testing. While testing the entire application on the target would be considered the "ideal" environment, it may not be feasible to exercise and gather requirements-based coverage and structural coverage in a fully integrated environment. Some testing may need to be performed on small isolated components in a simulated environment, while others will need to be run on the real hardware or as part of software integration or hardware/software testing.

4.1.1 Low-Level Testing Environment

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 1) must be generated. Ideally, this should be done automatically with a tool that offers an intuitive and simple approach for defining test scenarios.

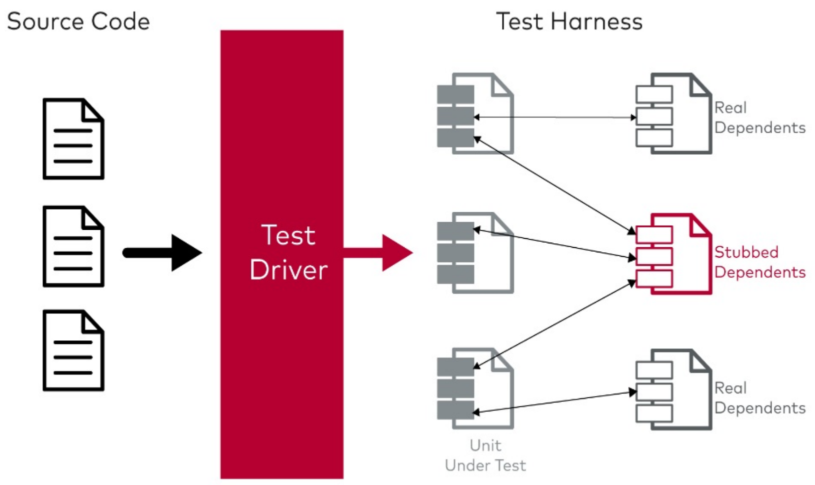
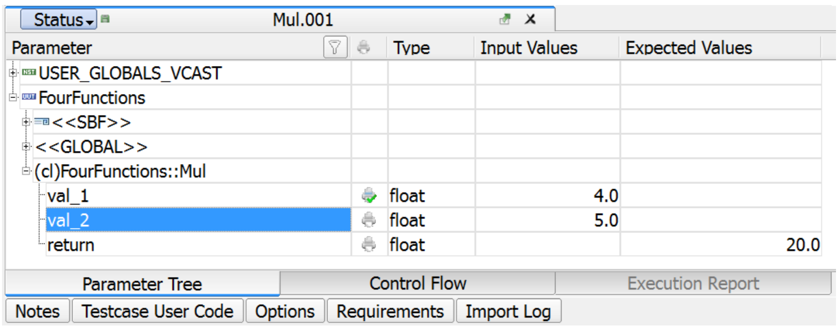


Figure 1: Unit & Integration Testing Framework

With the growing need for code reuse, it is very likely the same unit of source code might be used in several configurations. Hence 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. An example of this can be seen in Figure 2.



Parameter	Type	Input Values	Expected Values
USER_GLOBALS_VCAST			
FourFunctions			
<<SBF>>			
<<GLOBAL>>			
(cl)FourFunctions::Mul			
val_1	float	4.0	
val_2	float	5.0	
return	float		20.0

Figure 2: Test case parameter editor tree

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.

VectorCAST fully supports testing on target or using a target instruction set 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 manner.

4.1.2 Software Integration Testing Test Environment

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 usual to use a native 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 be optionally 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 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 dependent 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 more robust simulation of the underlying hardware is required to correctly verify the software functionality in this case. In this case, CANoe provides the facility to simulate external bus interfaces and peripherals that exist on that bus. The simulated bus and simulated peripherals can be simulated using MathWorks Simulink models, or programmatically using a scripting language CAPL. An example of this is shown in Figure 3.

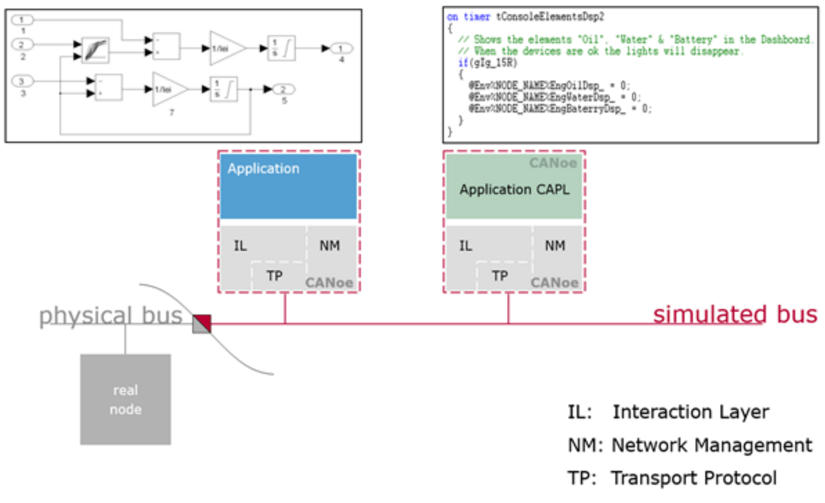

```
on timer tConsoleElementsDsp2
{
  // Shows the elements "Oil", "Water" & "Battery" in the Dashboard.
  // When the devices are ok the lights will disappear.
  if(gIg_15R)
  {
    @Env%NODE_NAME%EngOilDsp_ = 0;
    @Env%NODE_NAME%EngWaterDsp_ = 0;
    @Env%NODE_NAME%EngBaterryDsp_ = 0;
  }
}
```

Figure 3: Software Integration Testing Test Environment

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 possibilities to define tests range from programming in various languages like Vector's 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. The finished test sequences are stored as test units and are then executed in CANoe.

Test scenarios that use CANoe can be designed visually using vTESTstudio. vTESTstudio, in a similar to VectorCAST, takes a data driven approach for the design of test cases. This once again allows easy reuse of test cases and configurations and systems evolve over time or are used in multiple variants or configurations. vTESTstudio provides a modern authoring tool as seen in Figure 4.

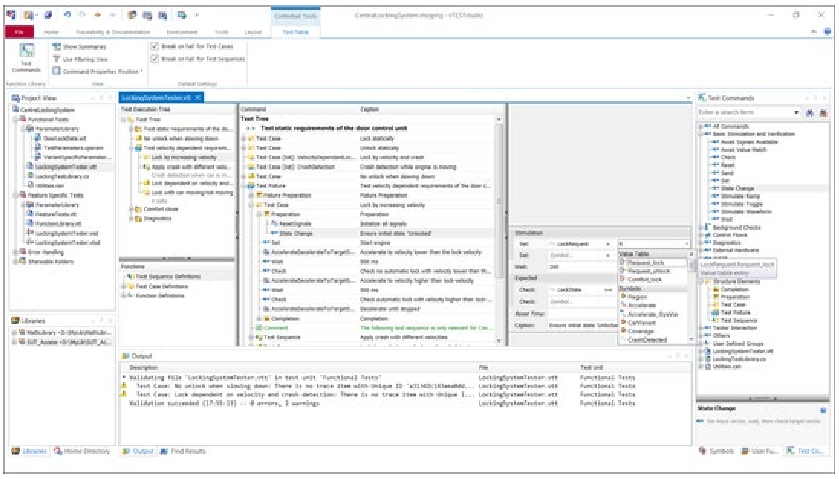

```
Description                                                                                   File                       Test Unit
• Validating file 'LockingSystemTester.vtt' in test unit 'Functional Tests'                   LockingSystemTester.vtt    Functional Tests
  Test Case: No unlock when slowing down: There is no trace item with Unique ID 'a31362c183aea0d0...   LockingSystemTester.vtt    Functional Tests
  Test Case: Lock dependent on velocity and crash detection: There is no trace item with Unique I...   LockingSystemTester.vtt    Functional Tests
  Validation succeeded (17:55:13) -- 0 errors, 2 warnings                                     LockingSystemTester.vtt    Functional Tests
```

Figure 4: vTESTstudio test case authoring tool

4.1.3 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 stimulation 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 extend or customize the simulation interfaces quickly and easily.

An example system to validate an LRU at this level can be setup using the VT System and CANoe (Figure 5). 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 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 once constructed a test system can be used for different LRUs.

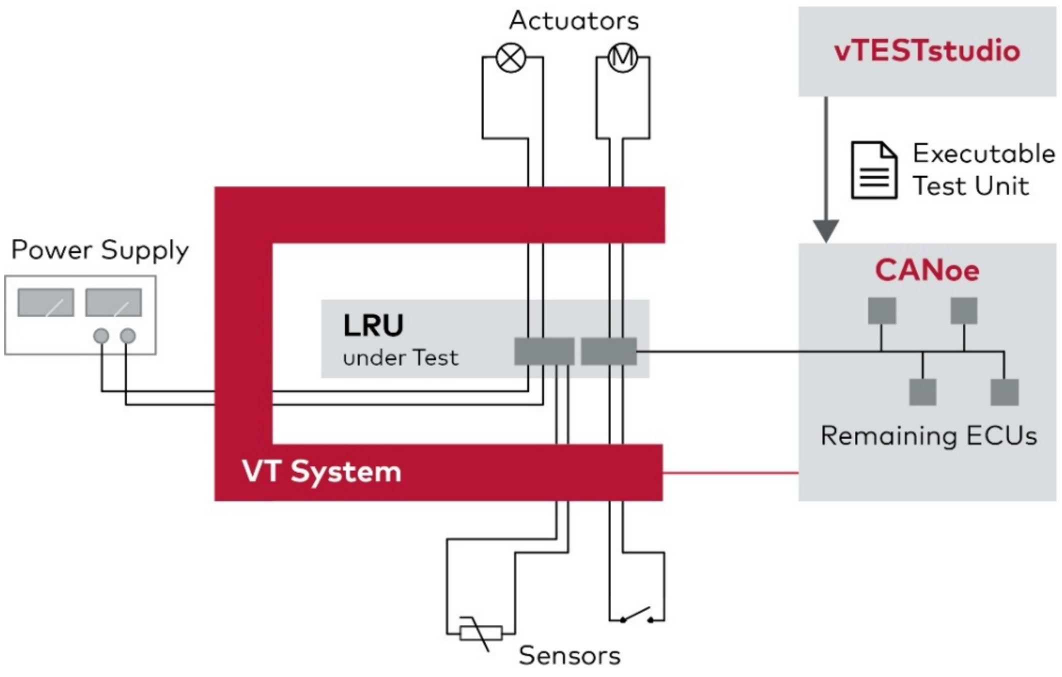


Figure 5: VT System setup

As discussed previously, the same methods of designing test scenarios in CANoe or vTESTstudio can be reused when working with VT System. This means any test scenarios that were developed using the software integration testing phase could be reused during the hardware/software integration testing phase.

4.1.4 Merging Test Environment Results Together

By using Vector's structural code coverage technology, it is possible to collect code coverage from all levels of testing like low-level, software integration and hardware/software integration. The ability to integrate the code coverage reporting with low-level testing and software integration tools like VectorCAST and with hardware/software integration tools like vTESTstudio, 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.

An additional benefit of Vector's code coverage technology is its ability to provide change based testing (CBT) support. When a change is made to the underlying software, the Vector coverage 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. We can see a visual of how this works in Figure 6.

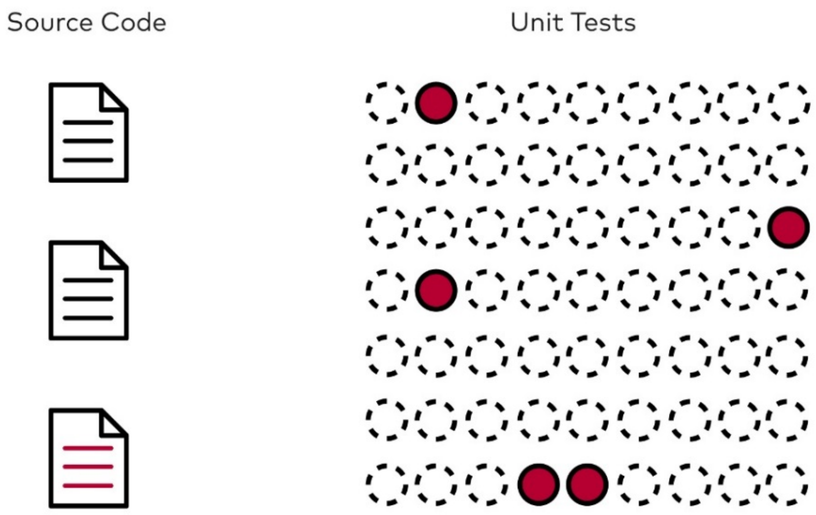


Figure 6: Change Based Testing visualization

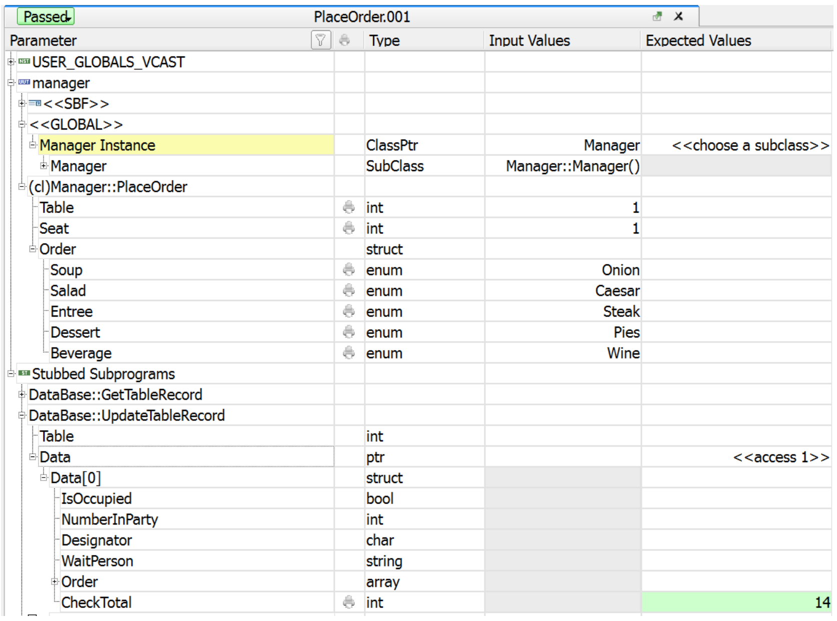
4.2 Section 6.4.2 – Requirements Based Test Selection

DO-178C specifies that the software verification should be “requirements based”, as opposed to source code based. Requirements based tests will require that testers or developers build the input data to exercise the code that will satisfy the requirement. These requirements based tests will take on two forms: *normal range test cases* and *robustness test cases*.

4.2.1 Section 6.4.2.1 – Normal Range Test Cases

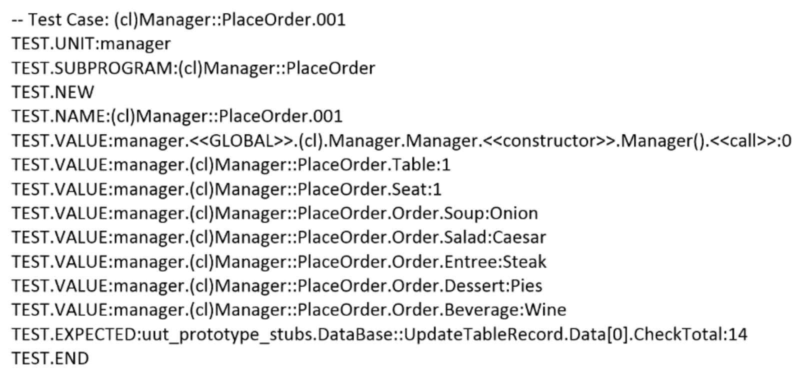
4.2.1.1 Low-Level Normal Range Test Cases

The objective of normal range tests is to demonstrate the ability of the software to respond to normal inputs and operating conditions. Specifically, real and integer input values should be exercised. VectorCAST provides the ability to set these values via the GUI or through a script. The following graphics show the GUI (Figure 7) and script (Figure 8) examples:



Parameter	Type	Input Values	Expected Values
USER_GLOBALS_VCAST			
manager			
<<SBF>>			
<<GLOBAL>>			
Manager Instance	ClassPtr	Manager	<<choose a subclass>>
Manager	SubClass	Manager::Manager()	
(cl)Manager::PlaceOrder			
Table	int	1	
Seat	int	1	
Order	struct		
Soup	enum	Onion	
Salad	enum	Caesar	
Entree	enum	Steak	
Dessert	enum	Pies	
Beverage	enum	Wine	
Stubbed Subprograms			
DataBase::GetTableRecord			
DataBase::UpdateTableRecord			
Table	int		
Data	ptr		<<access 1>>
Data[0]	struct		
IsOccupied	bool		
NumberInParty	int		
Designator	char		
WaitPerson	string		
Order	array		
CheckTotal	int		14

Figure 7: VectorCAST test case editor



```
-- Test Case: (cl)Manager::PlaceOrder.001
TEST.UNIT:manager
TEST.SUBPROGRAM:(cl)Manager::PlaceOrder
TEST.NEW
TEST.NAME:(cl)Manager::PlaceOrder.001
TEST.VALUE:manager.<<GLOBAL>>.(cl).Manager.Manager.<<constructor>>.Manager().<<call>>:0
TEST.VALUE:manager.(cl)Manager::PlaceOrder.Table:1
TEST.VALUE:manager.(cl)Manager::PlaceOrder.Seat:1
TEST.VALUE:manager.(cl)Manager::PlaceOrder.Order.Soup:Onion
TEST.VALUE:manager.(cl)Manager::PlaceOrder.Order.Salad:Caesar
TEST.VALUE:manager.(cl)Manager::PlaceOrder.Order.Entree:Steak
TEST.VALUE:manager.(cl)Manager::PlaceOrder.Order.Dessert:Pies
TEST.VALUE:manager.(cl)Manager::PlaceOrder.Order.Beverage:Wine
TEST.EXPECTED:uut_prototype_stubs.DataBase::UpdateTableRecord.Data[0].CheckTotal:14
TEST.END
```

Figure 8: VectorCAST test case data description in a script format

For time-related functions, multiple iterations of the code should be performed to ensure the correct characteristics of the function under test.

VectorCAST provides a simple and convenient way to iterate tests over time. This functionality is called “compound testing”. It allows a tester to execute a single test many times or multiple tests over time while ensuring that the data remains persistent across all executions.

For requirements expressed by logic equations, it may be necessary to perform Modified Condition/Decision Coverage (MC/DC). Vector’s testing tools code coverage support provides MC/DC levels of coverage and the associated equivalence pair matrices.

4.2.1.2 Section 6.4.2.2 – Robustness Test Cases

The objective of robustness test cases is to demonstrate the ability of the software to respond to abnormal inputs and conditions. These types of tests aim at removing potential software errors at parameter limits or boundaries based on the parameter’s type.

The best way to achieve this goal is through low-level whitebox testing to permit inputting these boundary limit values directly to functions. There are some “special cases” that should be correctly tested. These include:

> Setting parameters to zero if they happen to divide another variable within the function

> Testing blank ASCII characters

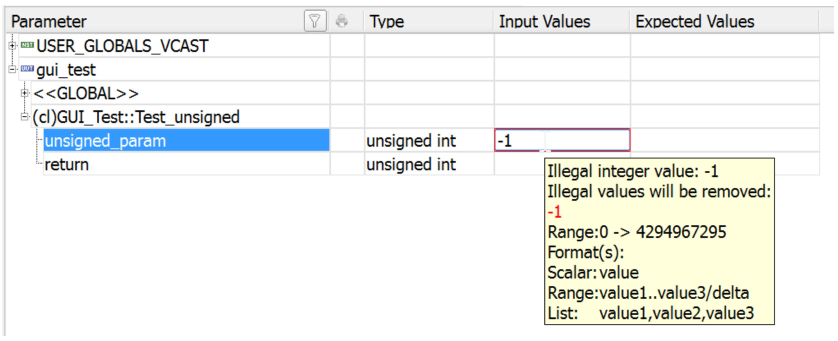
> Testing an empty stack or list element

> Testing a null matrix

> Testing a zero-table entry

> Testing the maximum and the minimum values of the type, and potentially the functional limits

> Testing values outside the boundaries (Figure 9)



Parameter	Type	Input Values	Expected Values
USER_GLOBALS_VCAST			
gui_test			
<<GLOBAL>>			
(cl)GUI_Test::Test_unsigned			
unsigned_param	unsigned int	-1	
return	unsigned int		

```
Illegal integer value: -1
Illegal values will be removed:
-1
Range:0 -> 4294967295
Format(s):
Scalar:value
Range:value1..value3/delta
List:    value1,value2,value3
```

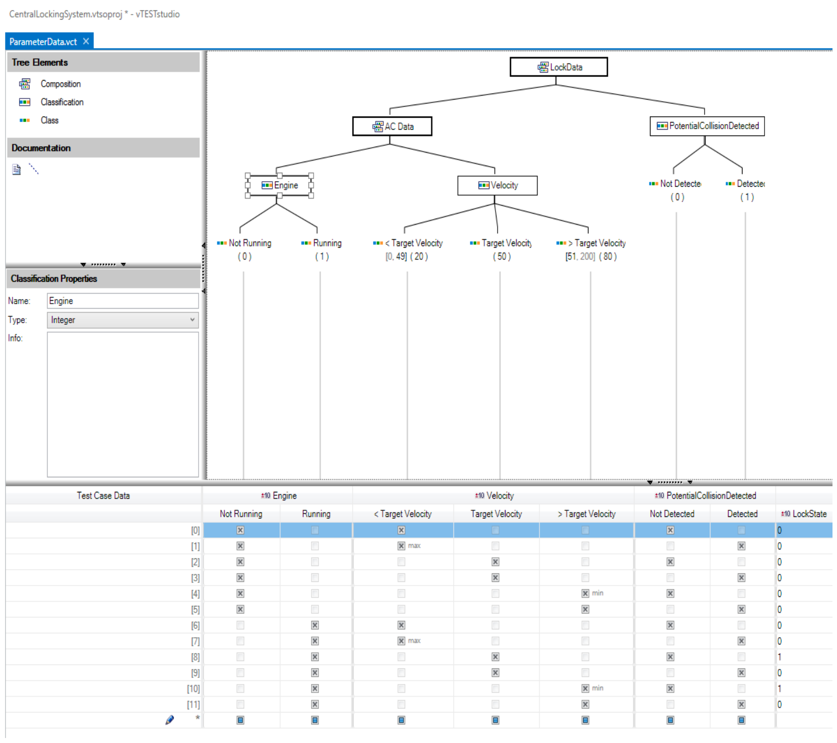
Figure 9: Defining out of bound test cases

Some test automation tools can automate the construction of these test cases even further by providing a way to automatically generate test cases that test all input values to their minimum, maximum and median values. These sorts of test cases are referred to as MIN-MID-MAX test cases.

The minimum and maximum values are determined by testing the range of every type present in the program on the target board or simulator. Thus, using the test automation tool on either the board or on a simulator will guarantee that the range of boundary values tested through automatically generated MIN-MID-MAX tests is valid for the system under test.

These tools can test special values as specified above, additionally, they can even test other values not directly mentioned, such as Not-A-Number (NAN), positive and negative infinity on floating-point variables, etc.

The use of a classification tree method can also greatly enhance the ability to ensure all robustness edge cases are tested within a software unit. The integrated classification tree editor in vTESTstudio provides a mechanism to specify the classification tree method test case data - in terms of test vectors. The graphical user interface (Figure 10) supports finding the relevant input data for a test. An automatic or manual combination of all crucial input values allows to efficiently define the minimum number of required test vectors.



Test Case Data	Engine		Velocity			PotentialCollisionDetected		
	Not Running	Running	< Target Velocity	Target Velocity	> Target Velocity	Not Detected	Detected	LockState
[0]	☒	☐	☒	☐	☐	☒	☐	0
[1]	☒	☐	☒ max	☐	☐	☐	☒	0
[2]	☒	☐	☐	☒	☐	☒	☐	0
[3]	☒	☐	☐	☒	☐	☐	☒	0
[4]	☒	☐	☐	☐	☒ min	☒	☐	0
[5]	☒	☐	☐	☐	☒	☐	☒	0
[6]	☐	☒	☒	☐	☐	☒	☐	0
[7]	☐	☒	☒ max	☐	☐	☐	☒	0
[8]	☐	☒	☐	☒	☐	☒	☐	1
[9]	☐	☒	☐	☒	☐	☐	☒	0
[10]	☐	☒	☐	☐	☒ min	☒	☐	1
[11]	☐	☒	☐	☐	☒	☐	☒	0
*	■	■	■	■	■	■	■	

Figure 10: Definition of test vectors by the editor for the classification tree method

4.2.2 Section 6.4.3 – Requirements Based Testing Methods

The requirements based testing methods consist of the following types:

> Hardware/Software Integration Testing (6.4.3a)
> Software Integration Testing (6.4.3b)
> Low-Level Testing (6.4.3c)

Vector supports all three levels of testing defined. For hardware/software integration testing, Vector’s code coverage support provides structural coverage capabilities for projects developing to Levels A, B, and C. This testing can be performed on target or with the use of a target simulator independent of Vector’s testing tools, or in combination with vTESTstudio, CANoe, and VT System.

For software integration and low-level testing, VectorCAST provides the ability to construct executable test harnesses automatically allowing the testing of individual components or a collection of components, while CANoe & vTESTstudio can be used for API testing. While structural coverage is gathered during these phases of testing and is aggregated to show the completeness of testing.

4.2.3 Section 6.4.4 – Test Coverage Analysis

Test coverage analysis involves verifying requirements based coverage and structural coverage analysis. The first step is to analyze the test cases to confirm that all requirements are associated with a specific test case. The second step verifies that the requirements-based tests exercise the code to the appropriate level.

Vector's code coverage analysis determines which lines of source code (statement), which branches of source code (branch), or which equivalence pairs (MC/DC) have been executed by one or more sets of test case data. The reports show the completeness of the test suite. By analyzing the untested code, it is easy to work backward designing test cases to test these portions of code.

Objectives:

> Test coverage for high-level requirements is achieved
> Test coverage for low-level requirements is achieved
> Test coverage of software structure to appropriate criteria is achieved
> Test coverage of software structure, both data coupling, and control coupling is achieved

4.2.3.1 Section 6.4.4.1 – Requirements Based Coverage Analysis

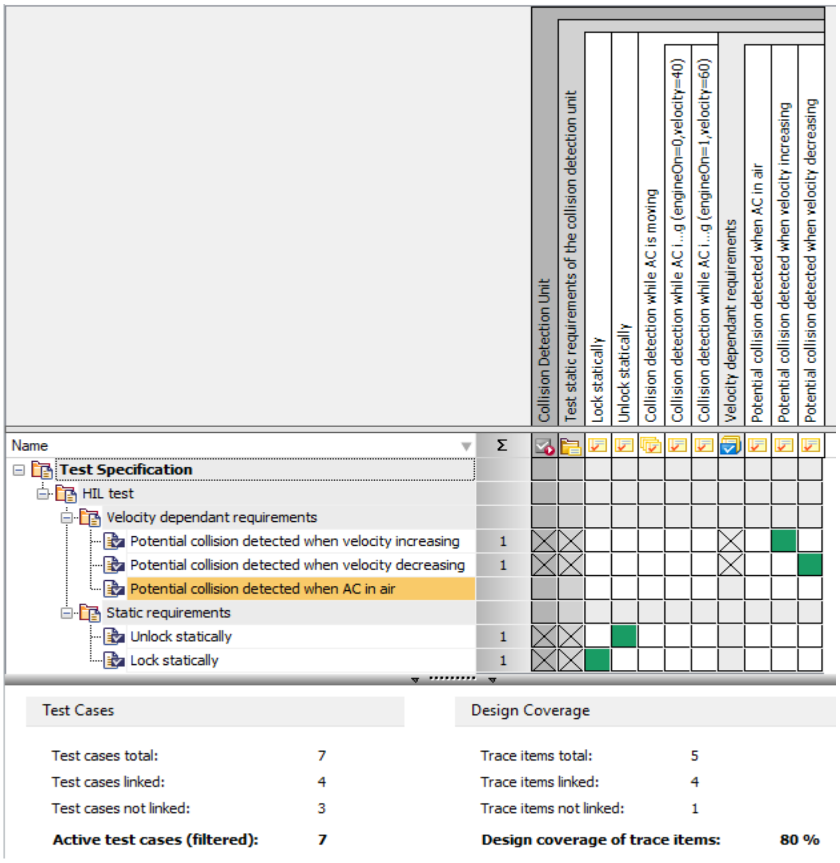
The Vector suite of testing tools offer the ability to link requirements with test cases to satisfy the requirements coverage analysis.

The requirements gateway permits the flow of data between a requirements management tool and the low-level, software integration and hardware/software testing tools from Vector's testing tools (i.e. VectorCAST and vTESTstudio). Through a simple and intuitive interface, developers can quickly and easily link requirements to their respective test cases.

Once test cases have been executed it is also possible to generate a traceability matrix between requirements and test cases as shown in Figure 11: Requirements traceability matrix. The matrix can be used for change impact analysis and requirements coverage reporting.

The user has full control over which test case attributes are passed back to the requirements database. Data such as the "Test Name" and "Test Result" {Pass | Fail | none} can be linked to user chosen attributes in the requirements database. The exchange of data is bi-directional and interactive between the Vector testing tools and the requirements management tool.

The imported data can be used by the test or requirements management tool to provide additional metrics reporting.



Name	Σ	Collision Detection Unit	Test static requirements of the collision detection unit	Lock statically	Unlock statically	Collision detection while AC is moving	Collision detection while AC i...g (engineOn=0,velocity=40)	Collision detection while AC i...g (engineOn=1,velocity=60)	Velocity dependant requirements	Potential collision detected when AC in air	Potential collision detected when velocity increasing	Potential collision detected when velocity decreasing
Test Specification												
HIL test												
Velocity dependant requirements												
Potential collision detected when velocity increasing	1	☒	☒						☒		🟩	
Potential collision detected when velocity decreasing	1	☒	☒						☒			🟩
Potential collision detected when AC in air												
Static requirements												
Unlock statically	1	☒	☒		🟩							
Lock statically	1	☒	☒	🟩								

Test Cases	
Test cases total:	7
Test cases linked:	4
Test cases not linked:	3
Active test cases (filtered):	**7**

Design Coverage	
Trace items total:	5
Trace items linked:	4
Trace items not linked:	1
Design coverage of trace items:	**80 %**

Figure 11: Requirements traceability matrix

4.2.3.2 Section 6.4.4.2 – Structural Coverage Analysis

The objective of this analysis is to determine which code structure has not been exercised by the requirements-based tests. Depending on the certification level, different levels of coverage will need to be achieved:

> Level A - 100% Statement + MCDC Coverage

> Level B - 100% Statement + Branch Coverage

> Level C - 100% Statement Coverage

> Level D-E - No mandatory requirement for Code Coverage

Understanding the Different Coverage Types

Structural Coverage – Statement Coverage

Statement coverage attempts to cover individual lines of code in a given unit. For instance, the following line would require a single test case to be covered.

```
if(i < 10 && j == 0 || k > 12)
```

It should be noted that a single test case evaluating to false at this line will not cover additional lines that may be contained within that ‘IF’ statement. Likewise, if this ‘IF’ statement has an ‘ELSE’ statement attached to it, a test case evaluating to true will not cover the contents of that ‘ELSE’ statement. However, either the true or false test case will cover the ‘IF’ statement per se.

Some unit test automation tools with automatic test case generation capabilities can help engineers devise test cases seeking to maximize statement coverage. For instance, automatically generated test cases based on the basis paths (the independent paths in the code) will often provide a high degree of statement coverage.

Structural Coverage – Branch Coverage

Branch coverage concentrates on the state of decision points, which can be either true or false. For instance, the following line of code would require two test cases to be covered – one where the ‘IF’ statement returns true, and another where it returns false.

if(i < 10 && j == 0 || k > 12)

To comply with DO-178C Level B, a code coverage tool should be able to produce both statement and branch levels of coverage during a single test execution.

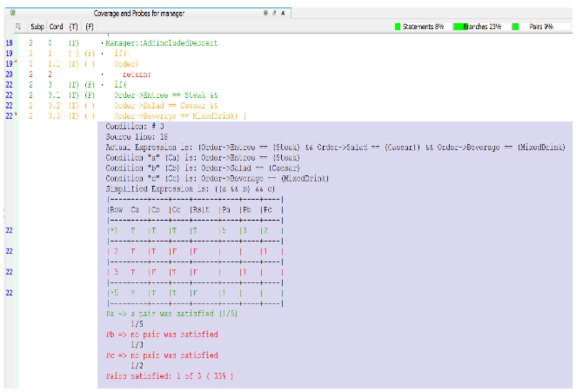
Structural Coverage – MC/DC (Modified Condition/Decision Coverage)

MC/DC requires the greatest number of tests to accomplish the coverage requirement. This level of coverage demonstrates that all sub-conditions that are part of a conditional statement can independently affect the outcome of the conditional statement itself. For instance, in the case of the following statement:

if(i < 10 && j == 0 || k > 12)

One should demonstrate that by changing the value of ‘i’ while keeping the value of other sub-conditions stable, the end value will change.

This task can be very arduous even for the most experienced engineer. However, this testing can be done efficiently by using a truth table. This can be automatically generated from the code and it should indicate clearly which test case pairs are required to achieve MC/DC coverage, and then flags which test cases and test case pairs have been provided as shown in Figure 12. The Vector coverage tools also support masking MC/DC Coverage.



```
18    2  0    (T)      Manager::AddIncludedDessert
19    2  1    ( ) (F)    if(
19*   2  1.1  (T) ( )    Order)
20    2  2                 return;
22    2  3    (T) (F)    if(
22    2  3.1  (T) (F)    Order->Entree == Steak &&
22    2  3.2  (T) ( )    Order->Salad == Caesar &&
22*   2  3.3  (T) ( )    Order->Beverage == MixedDrink) |
                      Condition: # 3
                      Source line: 18
                      Actual Expression is: (Order->Entree == (Steak) && Order->Salad == (Caesar)) && Order->Beverage == (MixedDrink)
                      Condition "a" (Ca) is: Order->Entree == (Steak)
                      Condition "b" (Cb) is: Order->Salad == (Caesar)
                      Condition "c" (Cc) is: Order->Beverage == (MixedDrink)
                      Simplified Expression is: ((a && b) && c)
                      |--------+----+----+------+----+----+----|
                      |Row  Ca  |Cb  |Cc  |Rslt  |Pa  |Pb  |Pc  |
                      |--------+----+----+------+----+----+----|
22                    |*1   T   |T   |T   |T     |5   |3   |2   |
                      |--------+----+----+------+----+----+----|
22                    | 2   T   |T   |F   |F     |    |    |1   |
                      |--------+----+----+------+----+----+----|
22                    | 3   T   |F   |T   |F     |    |1   |    |
                      |--------+----+----+------+----+----+----|
22                    |*5   F   |T   |T   |F     |1   |    |    |
                      |--------+----+----+------+----+----+----|
                      Pa => a pair was satisfied (1/5)
                            1/5
                      Pb => no pair was satisfied
                            1/3
                      Pc => no pair was satisfied
                            1/2
                      Pairs satisfied: 1 of 3 ( 33% )
```

Figure 12: MC/DC Truth Table

Aggregating Coverage across test environments

The Vector code coverage functionality allows you to gauge the effectiveness of your test efforts by identifying which areas of an application were exercised during requirements-based testing. This provides a convenient way to analyze the completeness of your tests, ensuring that applications are not released with untested code. Vector's code coverage tool allows you to analyze any portion of your application, or the entire application at once. For each file that is analyzed, The Vector code coverage view shown in Figure 13 creates a source-viewer containing the following information:

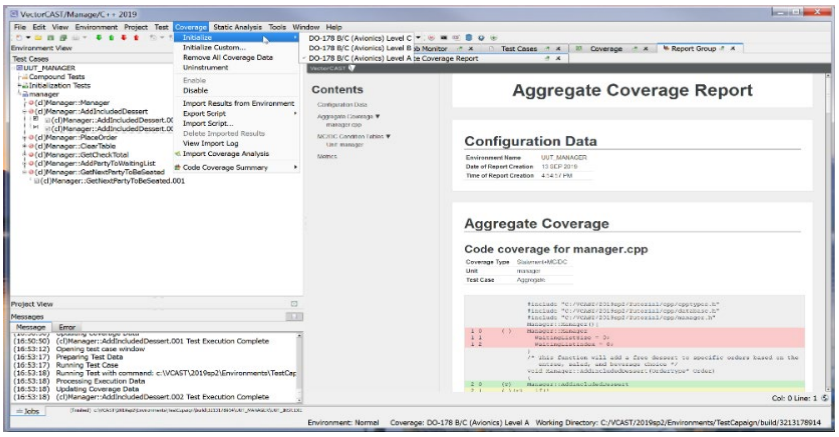
> Coverage summary provides a color-coded view of your source code that identifies code that is completely covered, partially covered, or uncovered

> Metrics summary provides a tabular list of code complexity and currently achieved source-code coverage for each subprogram

> Basis path analysis shows all control paths for each subprogram

> Modified Condition/Decision Coverage (MC/DC) for the RTCA DO-178B standard for Level A flight software

VectorCAST can measure code coverage directly when it is responsible for the definition and execution of the test cases as we have discussed for low-level testing, or it can easily be integrated with a VT System or CANoe or other tools like the workflow we have discussed in software integration testing or hardware/software testing.



Aggregate Coverage Report

Configuration Data

Environment Name	UUT_MANAGER
Date of Report Creation	13 SEP 2019
Time of Report Creation	4:54:57 PM

Aggregate Coverage

Code coverage for manager.cpp

Coverage Type	Statement+MC/DC
Unit	manager
Test Case	Aggregate

```
(16:50:50) (cl)Manager::AddIncludedDessert.001 Test Execution Complete
(16:53:12) Opening test case window
(16:53:17) Preparing Test Data
(16:53:17) Running Test Case
(16:53:18) Running Test with command: c:\VCAST\2019sp2\Environments\TestCap
(16:53:18) Processing Execution Data
(16:53:18) Updating Coverage Data
(16:53:18) (cl)Manager::AddIncludedDessert.002 Test Execution Complete
```

Environment: Normal Coverage: DO-178 B/C (Avionics) Level A Working Directory: C:/VCAST/2019sp2/Environments/TestCapaign/build/3213178914

Figure 13: Vector code coverage view

4.2.3.3 Section 6.4.4.3 – Structural Coverage Analysis Resolution

The objective of this analysis is to determine which code structure has not been exercised by the requirements based tests and determine additional software verification process activity.

It is very common for small portions of applications to be difficult or impossible to test. In these cases, developers in regulated industries are required to perform analysis of the uncovered code, and to document that analysis as part of their requirement to achieve 100% structural coverage. VectorCAST's Cover By Analysis (CBA) feature allows users to do this analysis within the Vector coverage views, and easily view the combined coverage metrics from test and analysis in a single report. Additionally, analysis can be imported from third party tools and shared across distributed teams, and the entire application life-cycle.

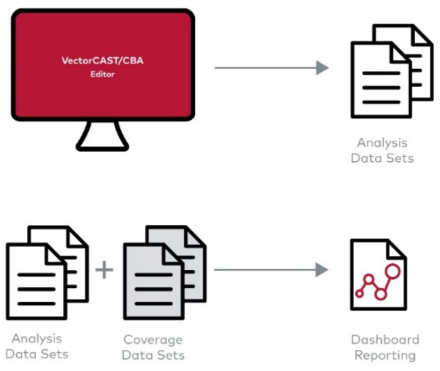


Figure 14: Coverage by analysis

4.2.4 Section 6.4.4.d Data Coupling and Control Coupling

Coupling verification is mandatory for safety-critical avionics software development according to RTCA DO-178C. The tools support the coupling requirements of DO-178C by using a combination of static analysis to identify the couples in a code base and runtime verification of the couples during application execution. Both the static analysis and runtime verification operate on software “components”. A component is a user-defined collection of one or more source files.

Coupling analysis intends to prove that the control and data flow between architectural components in the implementation match what was intended by the design and to prove that these flows have been tested. DO-178C requires applicants to identify couples in the design and to verify that those couples, and only those couples, exist in the implementation. Additionally, it requires applicants to verify that the couples have been exercised during functional requirements testing. VectorCAST’s component report, couples report, and coupling coverage report provide this proof (Figure 15 and 16).

VectorCAST’s Coupling feature provides data coupling and control coupling verification for C and C++ source files -- mandatory for safety-critical avionics software development according to DO-178B/C. The VectorCAST’s Coupling feature works in conjunction with the Vector code coverage package.

> Automated coupling instrumentation
> Works with your existing source code, build scripts, and tests
> Off-the-shelf reports to comply with DO-178C requirements
> Customizable component definitions

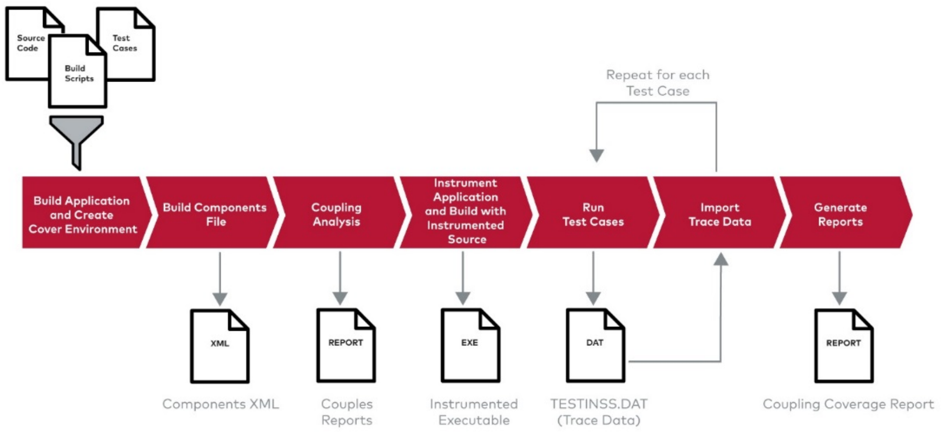
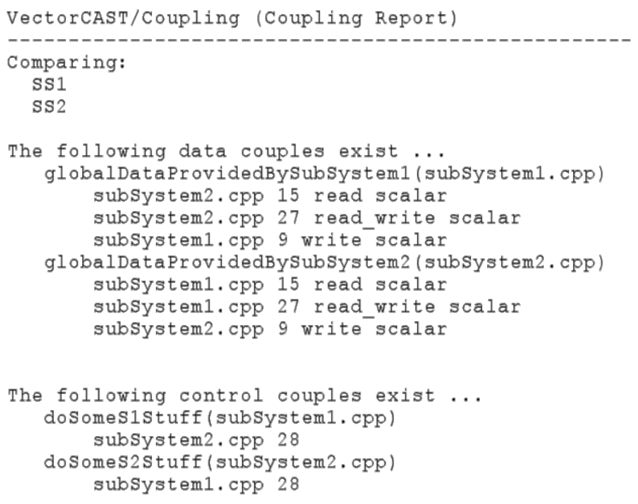


Figure 15: VectorCAST/Coupling workflow



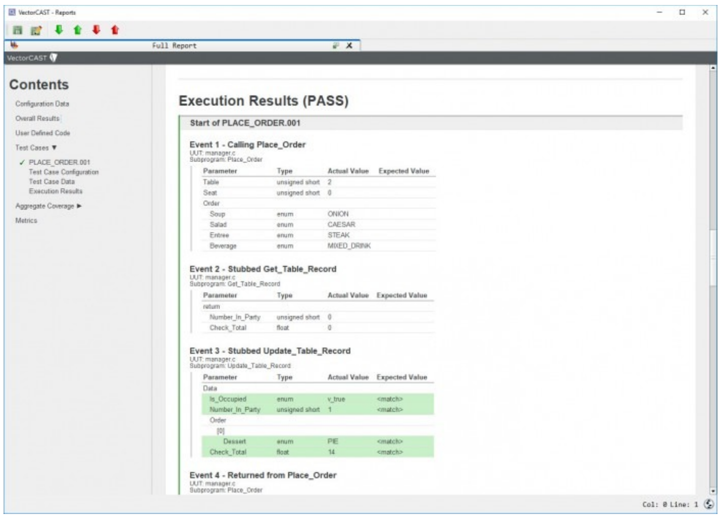
```
VectorCAST/Coupling (Coupling Report)
---------------------------------------------------
Comparing:
  SS1
  SS2

The following data couples exist ...
   globalDataProvidedBySubSystem1(subSystem1.cpp)
       subSystem2.cpp 15 read scalar
       subSystem2.cpp 27 read_write scalar
       subSystem1.cpp 9 write scalar
   globalDataProvidedBySubSystem2(subSystem2.cpp)
       subSystem1.cpp 15 read scalar
       subSystem1.cpp 27 read_write scalar
       subSystem2.cpp 9 write scalar


The following control couples exist ...
   doSomeS1Stuff(subSystem1.cpp)
       subSystem2.cpp 28
   doSomeS2Stuff(subSystem2.cpp)
       subSystem1.cpp 28
```

Figure 16: VectorCAST/Coupling report

4.2.5 Section 6.4.5 Reviews and Analyzes of Test Cases, Procedures, and Results

The requirement for this section is to review and analyze the test cases, test procedures, and test results.

The Vector testing tools allow test case specifications, execution results and structural code coverage reports to be easily exported in HTML or text formats for independent review and record keeping. The exportable reports can be customized to fit within organizational and corporate templates for the end user. An example of this is shown in Figure 17.



Execution Results (PASS)

Start of PLACE_ORDER.001

Event 1 - Calling Place_Order
UUT: manager.c
Subprogram: Place_Order

Parameter	Type	Actual Value	Expected Value
Table	unsigned short	2	
Seat	unsigned short	0	
Order			
Soup	enum	ONION	
Salad	enum	CAESAR	
Entree	enum	STEAK	
Beverage	enum	MIXED_DRINK	

Event 2 - Stubbed Get_Table_Record
UUT: manager.c
Subprogram: Get_Table_Record

Parameter	Type	Actual Value	Expected Value
return			
Number_In_Party	unsigned short	0	
Check_Total	float	0	

Event 3 - Stubbed Update_Table_Record
UUT: manager.c
Subprogram: Update_Table_Record

Parameter	Type	Actual Value	Expected Value
Data			
Is_Occupied	enum	v_true	<match>
Number_In_Party	unsigned short	1	<match>
Order			
[0]			
Dessert	enum	PIE	<match>
Check_Total	float	14	<match>

Event 4 - Returned from Place_Order
UUT: manager.c
Subprogram: Place_Order

Figure 17: Reporting view

4.3 Section 6.5 Software Verification Process Traceability

The requirement for this section is to demonstrate bi-directional traceability between requirements and test cases, test cases, and test procedures and test procedures and test results.

The Vector requirements gateway provides traceability between software requirements, test cases, and code coverage. It permits the bi-directional flow of data between a test or requirements management tool and the Vector testing tools.

The Vector requirements gateway provides visibility to ensure all requirements that are being tested, and if they have been successfully completed. It is integrated with the most popular requirements tools and satisfies the traceability requirements of regulated industry standards.

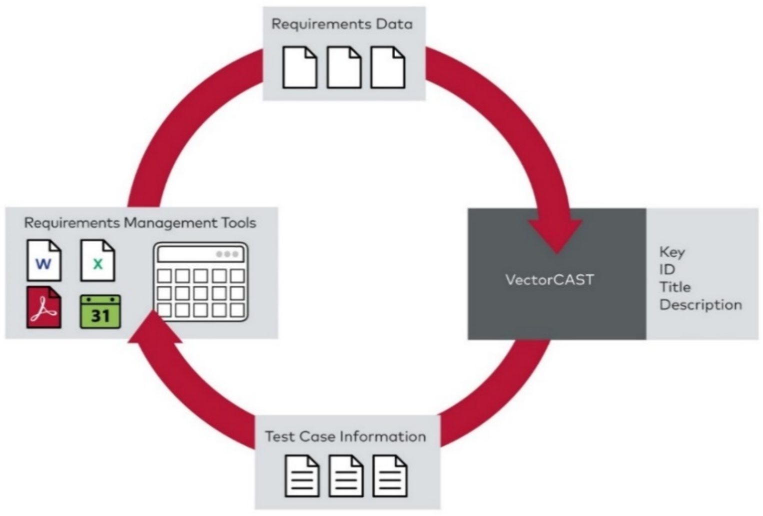


Figure 18: Requirements Gateway Workflow

4.4 Section 6.6 Verification of Parameter Data Items

Test cases developed using the Vector testing tools can easily be reused in different configurations for the LRU under test. Additionally, test cases can be set up to be conditional on a specific configuration, and only enabled when this specific configuration is under test. Conditional test cases remain part of the overall test case suite and are only activated when their associated configuration is under test.

5 Section 12 - Additional Considerations

This section in the standard provides guidance for additional consideration where “objectives and/or activities may replace, modify, or add to some or all of the objectives and/or activities defined in the rest of the DO-178C document”.

5.1 Section 12.1.3 – Change of Application or Development Environment

The use and modification of a previously developed application may involve a new development environment, compiler, target processor, or integration with other software other than that used in the original.

If a different compiler or different set of compiler options are used, resulting in different object code, the results from a previous software verification activity may not be valid for the new application.

5.1.1 Target Processor or Compiler Changes

It is important to note that all tests developed with VectorCAST should be considered non-target specific. The VectorCAST tests correspond to the data used by the application, not the target processor itself. When an application is ported to a new target processor, the VectorCAST tests can simply be re-executed in the new target environment.

This same approach is taken when a new compiler is used to build the application. The regression test mechanism in VectorCAST will re-build all executable test harnesses, invoking the new cross compiler, and all existing tests will be re-executed to verify the test results in the new environment. For new target processors or compiler environments, it may be necessary to re-qualify the VectorCAST tools for the new environment.

Using VectorCAST allows developers to import developed VectorCAST/C++ and VectorCAST/Ada test environments into regression test suites, providing a single point-of-control for all unit and integration test activities. As the VectorCAST test cases are target hardware and compiler agnostic, it is possible to create configurations in the VectorCAST Enterprise editions for the specific versions of compilers, linkers, debuggers, and target hardware. Once these configurations have been defined, it is possible to drag and drop the test suites into these configurations and rerun them (Figure 19).

VectorCAST provides the following testing infrastructure features:

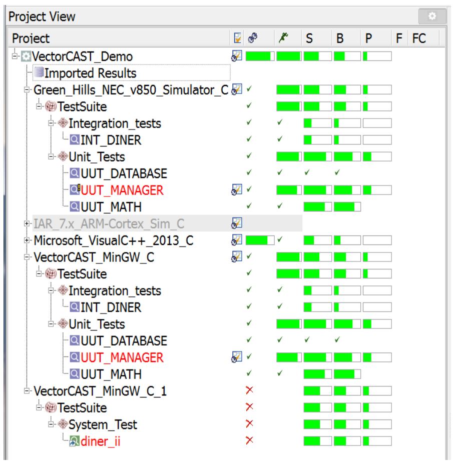
> The ability to organize thousands of test cases into logical groups that conform to the structure of your application

> The ability to determine the pass/fail status of each test case

> The ability to determine code coverage on a per function basis

> The ability to run the same test against multiple configurations of the application

> Command line and Python API interfaces to all of these features



```
Project View
Project                                  S   B   P   F  FC
VectorCAST_Demo
  Imported Results
  Green_Hills_NEC_v850_Simulator_C
    TestSuite
      Integration_tests
        INT_DINER
      Unit_Tests
        UUT_DATABASE
        UUT_MANAGER
        UUT_MATH
  IAR_7.x_ARM-Cortex_Sim_C
  Microsoft_VisualC++_2013_C
  VectorCAST_MinGW_C
    TestSuite
      Integration_tests
        INT_DINER
      Unit_Tests
        UUT_DATABASE
        UUT_MANAGER
        UUT_MATH
  VectorCAST_MinGW_C_1
    TestSuite
      System_Test
        diner_ii
```

Figure 19: VectorCAST Test Suite Tree for different compilers

vTESTstudio uses the message databases from the interface description document as a definition of the interfaces accessible in the tests. This allows the usage of test cases defined in vTESTstudio to be hardware and compiler agnostic. This also allows quick and easy reuse of test cases defined in vTESTstudio across various configurations.

5.1.2 Section 12.2.1 Tool Qualification

As stated in RTCA DO-178C Section 12.2.1, any tool that eliminates, reduces or automates any of the processes outlined within RTCA DO-178C without its output being verified as specified in RTCA DO-178C Section 6, must be qualified.

Vector's code coverage technology is used to automate the collection of structural coverage data during requirements based testing so that the structural coverage analysis can be performed, as described in RTCA DO-178C Section 6.4.4.2. Vector's code coverage technology must be qualified for this reason. The methodology for qualifying Vector's code coverage technology is covered under the VectorCAST Tool Qualification process.

5.1.2.1 VectorCAST Tool Qualification Level and Criteria

Before applying the requirements of DO-330/ED-215 to the tool qualification process, the first step is to establish the tool qualification level (TQL) and Criteria for the project. To qualify the tool, VectorCAST is classified as a Criteria 3 tool. The justification for this classification is as follows:

According to the guidance provided for selecting the tool qualification level (TQL) of a tool in RTCA DO-178C Section 12.2.2, VectorCAST can be categorized as Criteria 3 by eliminating its categorization as Criteria 1 and 2.

> VectorCAST is not a Criteria 1 tool because its output is not part of the airborne software and therefore cannot insert an error
> VectorCAST is not a Criteria 2 tool because

1. It cannot be used to justify the elimination or reduction of any verification process other than the collection of structured coverage data, and
2. It cannot be used to justify the elimination of reduction of any development processes

Therefore, VectorCAST is categorized as a Criteria 3 tool because it fits the criteria that it could fail to detect an error within the scope of its intended use. Criteria 3 tools are considered TQL-5 for all Software Levels according to RTCA DO-178C Table 12-1.

vTESTstudio, CANoe, and VT System are deployed as integrated solutions as part of the HIL testing environment. While the products have been used in the certification of safety-critical systems, the certification of the tools needs to be evaluated on a project by project basis and will be out of the scope of this document.

5.1.3 RTCA DO-330 Required Data Items

VectorCAST is a COTS tool which is developed independently of any software project and is intended to be utilized by multiple users. As such, Vector adheres to the guidance pertaining to the approach to qualifying COTS tools provided in RTCA DO-330 Section 11.3, specifically the activities designated to the tool developer, as described in RTCA DO-330 Section 11.3.2, which states that the following data items should be provided by the tool developer:

> Developer-TOR (RTCA DO-330 Section 11.3.2.1)
> Developer-TQP (RTCA DO-330 Section 11.3.2.2)
> Developer-TCI (RTCA DO-330 Section 11.3.2.3)
> Developer-TAS (RTCA DO-330 Section 11.3.2.4)

Additionally, by cross-referencing RTCA DO-330 Table 11-1 with the tables in Annex A of RTCA DO-330 we can see that these additional data items should also be provided by the tool developer for a TQL-5 tool:

> Tool Operational Verification and Validation Cases and Procedures
> Tool Operational Verification and Validation Results
> Tool Configuration Management Records
> Tool Quality Assurance Records (can be part of the Software Quality Assurance Records for TQL-5)

These data items provide the user with insight into the tool developer's development life cycle and a separation of objectives and activities for the tool developer and tool user.

It is necessary for the tool developer to provide these data items for these reasons, and to provide visibility to the user and the certifying authority into the tool operational development process, the tool development process and the related verification processes. The tool user would not be able to provide this information, as they do not have access to the tool developer´s internal development process.

Item	Description
Developer-TOR	See RTCA DO/330 Section 11.3.2.1
Developer-TQP	See RTCA DO/330 Section 11.3.2.2
Developer-TCI	See RTCA DO/330 Section 11.3.2.3
Developer-TAS	See RTCA DO/330 Section 11.3.2.4
VectorCAST TQD	Contains the Tool Operational Verification and Validation Cases and Procedures and Results
VectorCAST Conformity Review	Verification of the Tool Life Cycle
Python-driven test suite	The actual Tool Operational Verification and Validation Cases and Procedures to be executed by the user

Table 2: Data items provided by the Tool Developer

The tool configuration management records and tool quality assurance records are not provided as part of the VectorCAST qualification kit due to the customer-sensitive information contained within these records and can be viewed during an on-site audit, and after signing an NDA.

The tool operational verification and validation cases and procedures and results are provided in the VectorCAST tool qualification document (TQD). The following sections describe each item in more detail.

5.1.3.1 VectorCAST Developer-TOR

The primary intent of this document is to provide the user with the applicable set of VectorCAST’s operationalr for the user’s operational environment so that the user may ascertain whether the tool meets their needs. The list of data included in this document is listed in RTCA DO-330 Section 11.3.2.1.

The user cannot produce this document on their own since these requirements define the functionality of the tool and are produced by the tool developer at the beginning of the tool’s development life cycle.

The developer-TOR is to be used instead of the TOR described in RTCA DO-330 Section 10.3.1.

5.1.3.2 VectorCAST Developer-TQP

The purpose of this document is to provide guidance to the user on how to qualify VectorCAST for use in their operational environment and set forth the proposed plan for doing so. The user would be unable to provide this documentation since it contains the tool life cycle information for the development of VectorCAST and describes how the preliminary qualification data should be used by the tool user.

This document contains all the same information as a TQP for a tool that would be developed by the user, as shown in RTCA DO-330 Section 10.1.2, but is limited to those activities that pertain to the tool developer.

5.1.3.3 VectorCAST Developer-TCI

The purpose of this document is to identify VectorCAST’s configuration management data, including specific configuration and version identifiers. This document cannot be produced by the user since is contains information that is only accessible to the tool developer.

This document contains all the same information as a TCI for a tool that would be developed by the user, as shown in RTCA DO-330 Section 10.1.11, but is limited to those activities that pertain to the tool developer.

5.1.3.4 VectorCAST Developer-TAS

The purpose of this document is to show that the tool qualification plan was followed and to highlight any discrepancies between the TQP and the actual qualification, including any problem reports (PR) raised as a result of the qualification effort. This document needs to be produced by the tool developer since it is the tool developer who will be tracking the PRs raised, if any, during the tool qualification process.

This document contains all the same information as a TAS for a tool that would be developed by the user, as shown in RTCA DO-330 Section 10.1.15, but is limited to those activities that pertain to the tool developer.

5.1.3.5 VectorCAST TQD

The purpose of this document is to provide the user with the tool operational verification and validation test cases and procedures, along with the tool operational verification and validation results. The test case provided in this document, and the Python-driven suite are to be executed in the user's operational environment and the results provided back to Vector for verification.

While technically the user could produce this document with the other information provided in the qualification kit, it is beneficial to the user for the tool developer to produce this document since they are better able to verify the test cases.

5.1.3.6 VectorCAST Conformity Review

The purpose of this document is to demonstrate that VectorCAST has undergone the appropriate quality assurance processes that the tool life cycle is complete, the tool life cycle data is complete, and the tool executable object code is controlled and can be generated. This satisfies the requirement for the tool conformity review as described in RTCA DO-330 Section 8.3.

5.1.3.7 VectorCAST Quality Assurance Records

These records contain the proof that VectorCAST has undergone its quality assurance process on the released version of the tool, and that any PRs raised and marked and resolved have been verified. These records contain user-specific information, so they are not provided as part of the qualification kit.

If the user wishes to view these records, they must schedule an on-site audit and sign a non-disclosure agreement due to the sensitive nature of the data contained within these records. This satisfies the requirement for the tool quality assurance records described in RTCA DO-330 Section 10.1.14.0.

6 Conclusion

As the complexities of avionics and ground based systems continues 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 functions 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 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 with the standards.

From commercial avionics to satellite systems to complex defense systems, Vector's test solutions are the proven standard for aerospace and defense industry leaders building software that must be reliable. For over 25 years, Vector has helped quality assurance organizations and software developers of safety-critical embedded systems deliver fully tested certified avionics applications. Whether your organization's codebase is considered legacy or is in initial development, Vector provides powerful solutions that enable companies to effectively test their products at using low-level testing (source code) to the high-level (physical interfaces) system testing and easily create automated continuous test environments, a goal of many companies.