VECTOR

Using VectorCAST for DO-178C Software Verification

Whitepaper | V3 2021-04

Table of Contents

1 Introduction

This whitepaper describes how the VectorCAST embedded software testing platform is used to satisfy the Software Verification Process objectives as defined in section 6.0 of the DO-178C standards, “Software Considerations in Airborne Systems and Equipment Certification”.

2 What are the DO-178B and DO-178C Standards

DO-178C is a revision to DO-178B in light of the experiences and information gathered with respect to developing software for avionics. Entitled DO-178C (ED-12C), it was completed in November 2011 and approved by the RTCA in December 2011. The new document became available for use in January 2012.

DO-178B was first published in December 1992 by RTCA, Incorporated. The document outlined 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

3 What is VectorCAST

The VectorCAST family of tools supports the capture and reporting of structural code coverage data at all levels prescribed by DO- 178C, including Level A, and the generation of all test artifacts required for a DO-178C audit. The VectorCAST product family consists of five complementary technologies:

VectorCAST/C++ VectorCAST/Ada	These tools automate the process of testing source modules written in C, C++ or Ada – for both unit testing and integration testing. Coverage data can be combined with VectorCAST/QA to achieve 100% coverage.
VectorCAST/QA	Generates code coverage artifacts during functional or system test of C, C++ or Ada.
VectorCAST RGW (Requirements Gateway)	Provides the ability to tag requirements to VectorCAST test cases.
VectorCAST RSP (Runtime Support Package)	Extends VectorCAST/C++ or VectorCAST/Ada to enable test execution for real-time applications on an embedded-target or in a simulator environment.
VectorCAST Enterprise Edition	Is used to automate the regression testing activities for all unit and integration tests.

The tools also support the creation and management of test cases to prove that the low-level software requirements have been tested.

4 How VectorCAST Supports DO-178C

The Software Verification Process objectives are defined in section 6.0 of the DO-178C standard.

4.1 Section 6.4 – Software Testing Process

This process specifies three types of testing:

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. VectorCAST/QA is used during this type of testing to capture the code coverage during execution of system or functional level test procedures.

Software Integration Testing

Software integration testing verifies the interrelationship of components. Testing at this level is performed with VectorCAST/C++ or VectorCAST/Ada 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.

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/C++ and VectorCAST/Ada are used during this testing phase.

Note: In each of these phases of test, structural coverage and requirements coverage are satisfied. A unique feature of VectorCAST is that the coverage gathered during each phase of test can be shared with another phase. This alleviates the need to perform low-level requirements-based testing when coverage has already been achieved at a higher level of test.

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. Testing may need to be performed on small, isolated components in a simulated environment.

VectorCAST fully supports testing on target or through the use of 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.

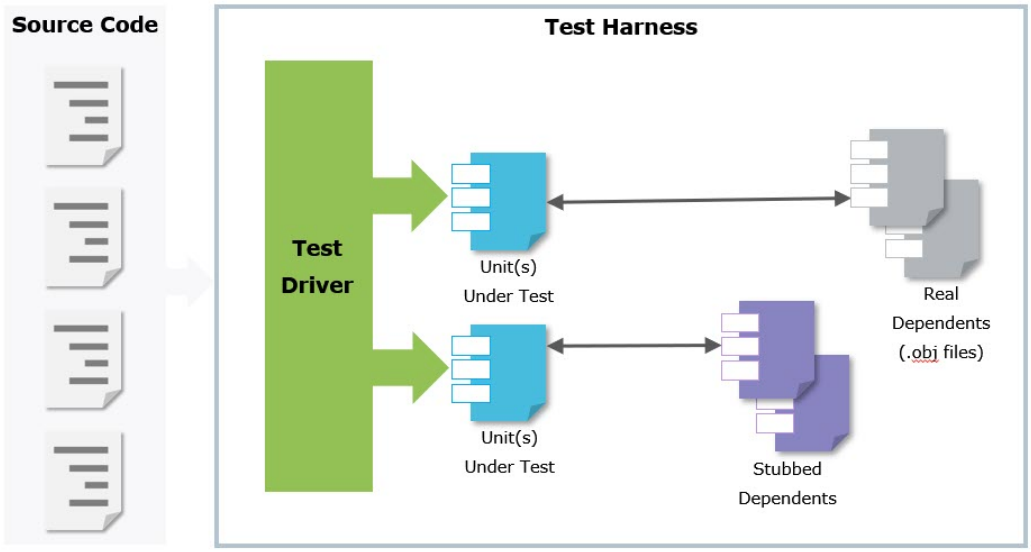


Figure 1: VectorCAST Test Harness for Unit Level Testing

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 actually 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.

Section 6.4.2.1 – 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 and script examples.

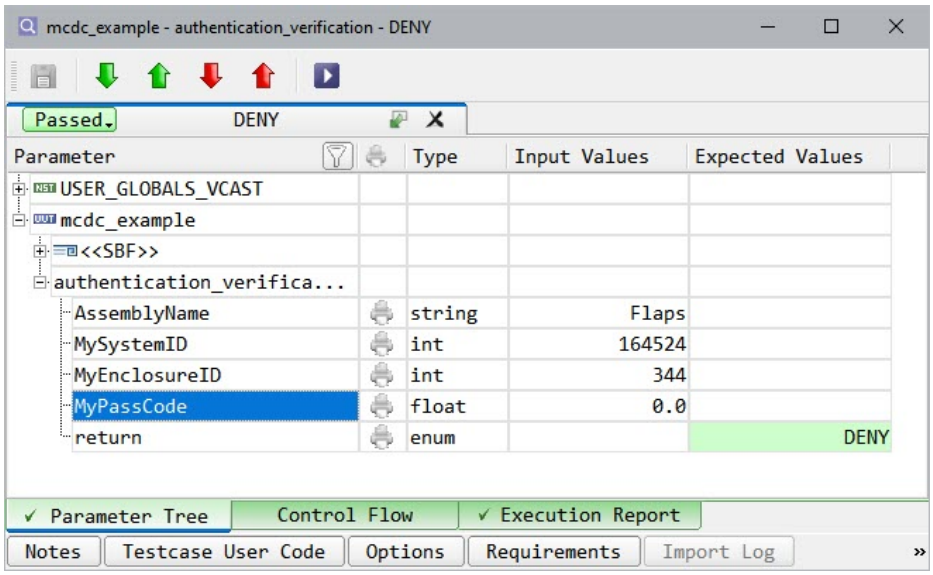


Figure 2: Normal Range Test Case

```
-- Test Case: DENY
TEST.UNIT:mcdc_example
TEST.SUBPROGRAM:authentication_verification
TEST.NEW
TEST.NAME:DENY
TEST.VALUE:mcdc_example.authentication_verification.AssemblyName:<<malloc 6>>
TEST.VALUE:mcdc_example.authentication_verification.AssemblyName:"Flaps"
TEST.VALUE:mcdc_example.authentication_verification.MySystemID:164524
TEST.VALUE:mcdc_example.authentication_verification.MyEnclosureID:344
TEST.VALUE:mcdc_example.authentication_verification.MyPassCode:0.0
TEST.EXPECTED:mcdc_example.authentication_verification.return:DENY
TEST.END
```

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). VectorCAST provides MC/DC levels of coverage and the associated equivalence pair matrices.

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. VectorCAST supports testing of “out of range” values or abnormal values for any type. This is controlled through an option to turn off normal range checking. The following graphic shows an example of setting an “out of range” value. Maximum value for this int variable is 2147482647, but with range checking disabled, that value can be set “out of range”.

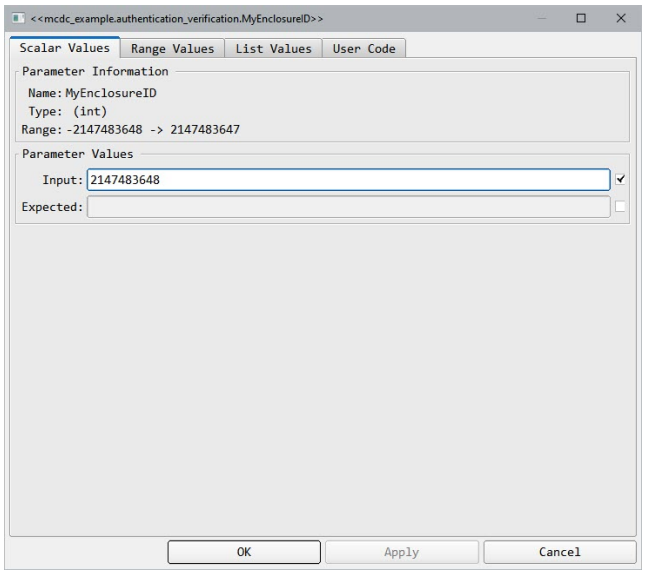


Figure 2: Robustness Test Case

Section 6.4.3 – Requirements-Based Testing Methods
The requirements-based testing methods consist of the following:

- Hardware/Software Integration Testing
- Software Integration Testing
- Low-Level Testing

VectorCAST supports all three levels of testing defined. For Hardware/Software integration testing, VectorCAST 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.

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. Structural coverage is gathered during these phases of testing and is aggregated to show the completeness of testing.

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.

VectorCAST 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 you the completeness of your test suite. By analyzing the untested code, you can easily work backward designing test cases to test these portions of your code.

Section 6.4.4.1 – Requirements-Based Coverage Analysis

The VectorCAST RGW (Requirements Gateway) allows the tagging of software requirements with test cases to satisfy the requirements coverage analysis.

The VectorCAST RGW permits the flow of data between a requirements management tool and the VectorCAST testing tools. Through a simple and intuitive interface, developers can quickly and easily link low-level requirements to VectorCAST test cases.

Once test cases have been executed the traceability metrics can be viewed from within the VectorCAST framework or the requirements management tool. The user has full control over which VectorCAST 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.

Section 6.4.4.2 – Structural Coverage Analysis

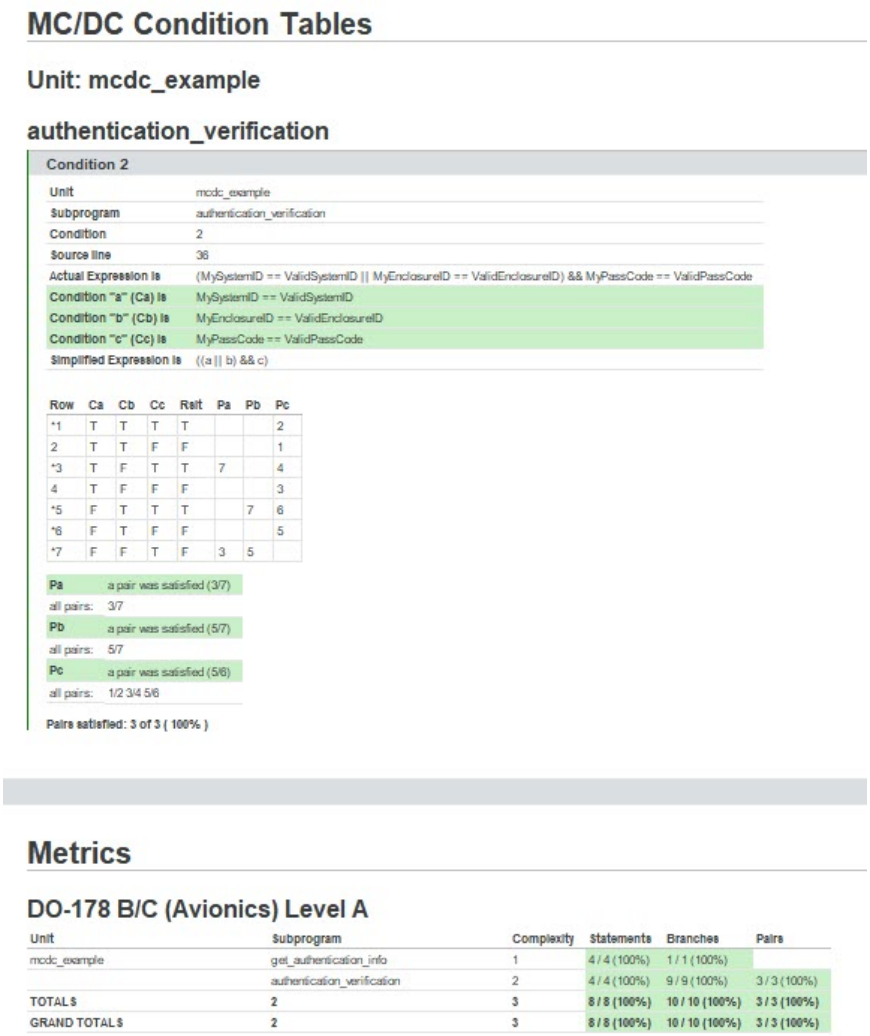
The objective of this analysis is to determine which code structure has not been exercised by the requirements-based tests.

VectorCAST 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. VectorCAST allows you to analyze any portion of your application, or the entire application at once. For each file that is analyzed, VectorCAST 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-178C standard for Level A flight software

MC/DC analysis shows that each sub-condition can independently affect the outcome of the entire condition. To prove this, you must be able to capture the values of the result of the condition as well as the value of each sub-condition. The VectorCAST Coverage Report captures this information in two formats: an annotated source listing, and the equivalence pair matrices for each Boolean condition.



MC/DC Condition Tables

Unit: mcdc_example

authentication_verification

Condition 2

Unit	mcdc_example
Subprogram	authentication_verification
Condition	2
Source line	36
Actual Expression is	(MySystemID == ValidSystemID || MyEnclosureID == ValidEnclosureID) && MyPassCode == ValidPassCode
Condition "a" (Ca) is	MySystemID == ValidSystemID
Condition "b" (Cb) is	MyEnclosureID == ValidEnclosureID
Condition "c" (Cc) is	MyPassCode == ValidPassCode
Simplified Expression is	((a || b) && c)

Row	Ca	Cb	Cc	Rslt	Pa	Pb	Pc
*1	T	T	T	T			2
2	T	T	F	F			1
*3	T	F	T	T	7		4
4	T	F	F	F			3
*5	F	T	T	T		7	6
*6	F	T	F	F			5
*7	F	F	T	F	3	5	

Pa	a pair was satisfied (3/7)
all pairs:	3/7
Pb	a pair was satisfied (5/7)
all pairs:	5/7
Pc	a pair was satisfied (5/6)
all pairs:	1/2 3/4 5/6

Pairs satisfied: 3 of 3 (100%)

Metrics

DO-178 B/C (Avionics) Level A

Unit	Subprogram	Complexity	Statements	Branches	Pairs
mcdc_example	get_authentication_info	1	4 / 4 (100%)	1 / 1 (100%)	
	authentication_verification	2	4 / 4 (100%)	9 / 9 (100%)	3 / 3 (100%)
TOTALS	**2**	**3**	**8 / 8 (100%)**	**10 / 10 (100%)**	**3 / 3 (100%)**
GRAND TOTALS	**2**	**3**	**8 / 8 (100%)**	**10 / 10 (100%)**	**3 / 3 (100%)**

Figure 3: MC/DC Equivalence Pair Matrix Report

```
•  if ( ((
   MySystemID == ValidSystemID) || (
   MyEnclosureID == ValidEnclosureID ))
                    && (
   MyPassCode == ValidPassCode ) )
Condition: # 2
Source line: 36
Actual Expression is: (MySystemID == ValidSystemID || MyEnclosureID == ValidEnclosureID) && MyPassCode == ValidPassCode
Condition "a" (Ca) is: MySystemID == ValidSystemID
Condition "b" (Cb) is: MyEnclosureID == ValidEnclosureID
Condition "c" (Cc) is: MyPassCode == ValidPassCode
Simplified Expression is: ((a || b) && c)
|----+----+----+----+------+----+----+----|
|Row |Ca  |Cb  |Cc  |Rslt  |Pa  |Pb  |Pc  |
|----+----+----+----+------+----+----+----|
|*1  |T   |T   |T   |T     |    |    |2   |
|----+----+----+----+------+----+----+----|
| 2  |T   |T   |F   |F     |    |    |1   |
|----+----+----+----+------+----+----+----|
|*3  |T   |F   |T   |T     |7   |    |4   |
|----+----+----+----+------+----+----+----|
| 4  |T   |F   |F   |F     |    |    |3   |
|----+----+----+----+------+----+----+----|
|*5  |F   |T   |T   |T     |    |7   |6   |
|----+----+----+----+------+----+----+----|
|*6  |F   |T   |F   |F     |    |    |5   |
|----+----+----+----+------+----+----+----|
|*7  |F   |F   |T   |F     |3   |5   |    |
|----+----+----+----+------+----+----+----|
Pa => a pair was satisfied (3/7)
      3/7
Pb => a pair was satisfied (5/7)
      5/7
Pc => a pair was satisfied (5/6)
      1/2 3/4 5/6
Pairs satisfied: 3 of 3 ( 100% )
```

Figure 4: MC/DC Annotated Source Listing

4.2 Structural Coverage for Level A

In all non- Level A cases, the structural coverage analysis is performed on the source code. For Level A, if the compiler generates object code that is not directly traceable to source code statements, then additional verification should be performed to establish the correctness of such generated code.

There are techniques that can be used to perform this additional verification, one of which is described in a separate whitepaper from Vector.

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.

4.3 Control and Data Coupling

VectorCAST/Coupling provides built-in analysis and instrumentation for data coupling and control coupling verification for C and C++ source files and is used in conjunction with an existing VectorCAST/QA project environment.

The tools support the Coupling requirements of DO-178C by using a combination of static analysis to identify the couples in a codebase and run-time verification of the couples during application execution.

Both the static analysis and run-time verification operate on software "components". A component is generally a user-defined collection of source files. DO-178C defines a component as "A self-contained part, combination of parts, subassemblies, or units that performs a distinct function of a system." (DO-178C ANNEX B, Glossary)

The intent of Coupling Analysis is 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. It also requires applicants to verify that the couples have been exercised during functional requirements testing. VectorCAST's Component Report and Coupling Coverage Report provide this proof. Coupling will detect the read and write of the data couples and capture calls to control couples.

For data couples, the goal is to capture the order of access to each couple. For control couples, the goal is to ensure all control couples are tested by recording each time the control couple is called.

5 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 rebuild 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.

6 Tool Qualification

Qualification of a software verification tool is necessary (note that the documentation requirements for a development tool are different) when verification processes defined in the documents are eliminated, reduced, or automated by the use of a software tool without its output being verified.

VectorCAST addresses the automation of the software verification process as defined by DO-178C. The qualification of a software tool is mandated by DO-178C section 12.2. The qualification is done on a project by project basis due to differences in compiler versions, target architectures, compiler patches, tool versions, etc.

In Section 12.2.1, DO-178C states: “A tool is qualified only for use on a specific system where the intentions to use the tool is stated in the Plan for Software Aspects of Certification (PSAC) that supports the system. If a tool previously qualified on one system is proposed for use on another system, it should be re-qualified within the context of the other system” ("DO-178C - Software Considerations in Airborne Systems and Equipment Certification" 84).

For DO-178C, a Tool Qualification Level (TQL) must first be determined to assess the impact of the tool in the software life cycle. Section 12.2.2 specifies “Criteria Levels” that define the impact of a tool. Three Criteria Levels are defined as:

- Criteria 1: A tool whose output is part of the airborne software and thus could insert an error.
- Criteria 2: A tool that automates verification process(es) that thus could fail to detect an error, and whose output is used to justify the elimination or reduction of:
  - Verification process(es) other than automated by the tool, or
  - Development process(es) that could have impact on airborne software.
- Criteria 3: A tool that, within the scope of its intended use, could fail to detect an error.

Based on these Criteria levels and the Software Level (A thru D), a TQL can be determined as defined by the table below:

Software Level	Criteria Level 1	2	3
A	TQL-1	TQL-4	TQL-5
B	TQL-2	TQL-4	TQL-5
C	TQL-3	TQL-5	TQL-5
D	TQL-4	TQL-5	TQL-5

Responsibility for qualifying software verification tools (like VectorCAST) is usually a joint effort between the prime contractor and the tool vendor. Based on a system safety assessment, failure condition categories are established. These failure condition categories determine the level of software integrity necessary for avionic safe operation.

For additional guidance on the tool qualification process, the RTCA DO-330 “Software Tool Qualification Considerations” document was developed in conjunction with DO-178C.

6.1 VectorCAST Tool Qualification Data Deliverables

To qualify VectorCAST as a software verification tool, Vector produces the following information for each project that the tool is to be used on:

DO-178C/ED-12C Tool Qualification Kit Contents

- Developer Tool Operational Requirements (TOR)
  - Tool's functionality in verifiable requirements
  - Project operational environment
  - Configuration management process
  - Method for attaining verification that VectorCAST has been satisfactorily tested against specified requirements
- Developer Tool Qualification Plan (TQP)
  - Guidance on qualification for operational environment
  - Tool life cycle information
- Developer Tool Configuration Index (TCI)
  - Identify Configuration Management data
  - Specific configuration and version identifiers
- Developer Tool Accomplishment Summary (TAS)
  - Highlight discrepancies between TQP and actual qualification
  - Problem Reports (PR)
- VectorCAST Tool Qualification Document (TQD)
  - Tool qualification test data and results
- VectorCAST Conformity Review
- Python-driven test suite
  - Designed to use your existing VectorCAST configuration

The tool qualification process normally includes interaction with the qualifying user as well as the associated certification authority and/or DER.

6.2 VectorCAST Tool Qualification Deliverables Process

Vector delivers the customer a baseline "draft" version of the qualification material and solicits comments. In addition, a test suite is delivered so the qualifying user can execute it in the exact environment used for the project. Vector incorporates the changes to the documents and associated tests, if applicable. Vector delivers version 1.0 of the Tool Qualification Documents for final approval.

VECTOR

Get More Information

Visit our website for:

> News
> Products
> Demo software
> Support
> Training classes
> Addresses

www.vector.com