White Paper

Automating Requirements-Based Testing for DO-178C

This paper examines how automatic test case generation can deliver significant cost savings, while satisfying the software verification objectives of DO-178C software levels C and above. We also explore what DO-178C specifies about requirements-based testing and the practicalities of automation. Finally, we will take a detailed look at combining use of an automated testing framework with techniques for efficient requirements management.

Contents

Copyright Notice

1 Introduction

One of the most important phases in the development of RTCA DO-178C software is the verification. This phase provides assurance that the objectives identified in DO-178C have been satisfied in accordance with requirements and that the subsequent actions, methods and results, are captured, demonstrable and traceable.

DO-178C specifies that all tests must be requirements-based. This form of testing is emphasized by the standard as the most effective way of finding errors (DO-178C section 6.4.2). Requirements based testing verifies that code complies only with its intended functionality and ensures that requirements are comprehensive and have been defined to a suitable level of detail.

However, the verification activities outlined in DO-178C section 6.0 (Software Verification Processes) are labour intensive and often add significant expense to a project. Whenever manual activities involve complex calculation, repetitive computation, or simply the transfer of information between existing technologies, there is scope for automation to add effectiveness and efficiency.

This paper will examine how automatic test case generation technology can deliver significant cost savings, when satisfying the software verification objectives of DO-178C software levels C and above. We will also explore what DO-178C specifies about requirements-based testing and the practicalities of automation. Finally, we will take a detailed look at combining the power of an automated testing framework with techniques for efficient requirements management.

The focus for this paper is primarily on low level requirements testing, because this area has the most scope for automation in meeting the process objectives.

For more information regarding requirements traceability and testing methods please visit – www.qa-systems.com

References

ACM Queue/Robert V. Binder, Bruno Legeard, and Anne Kramer. 2015. *Model-based Testing: Where Does It Stand?*. [ONLINE] Available at: https://queue.acm.org/detail.cfm?id=2723708. [Accessed 22 May 2018].

Microsoft/Sergio Mera, Yiming Cao. 2013. *Model Based Testing - An Introduction to Model-Based Testing and Spec Explorer*. [ONLINE] Available at: https://msdn.microsoft.com/en-us/magazine/dn532205.aspx. [Accessed 22 May 2018].

QA Systems GmbH. 2017. *Cantata Standard Briefing DO-178C*. [ONLINE] Available at: https://www.qa-systems.com/resources/detail/cantata-standard-briefing-do-178c/. [Accessed 17 May 2018].

Ralf Gerlich, Rainer Gerlich, Thomas Boll, Johannes Mayer. 2007. *Evaluation of Auto-Test Generation Strategies and Platforms*. [ONLINE] Available at: http://www.bsse.biz/pdfs/papers/DASIA2007-TestStrategies-paper.pdf. [Accessed 24 May 2018].

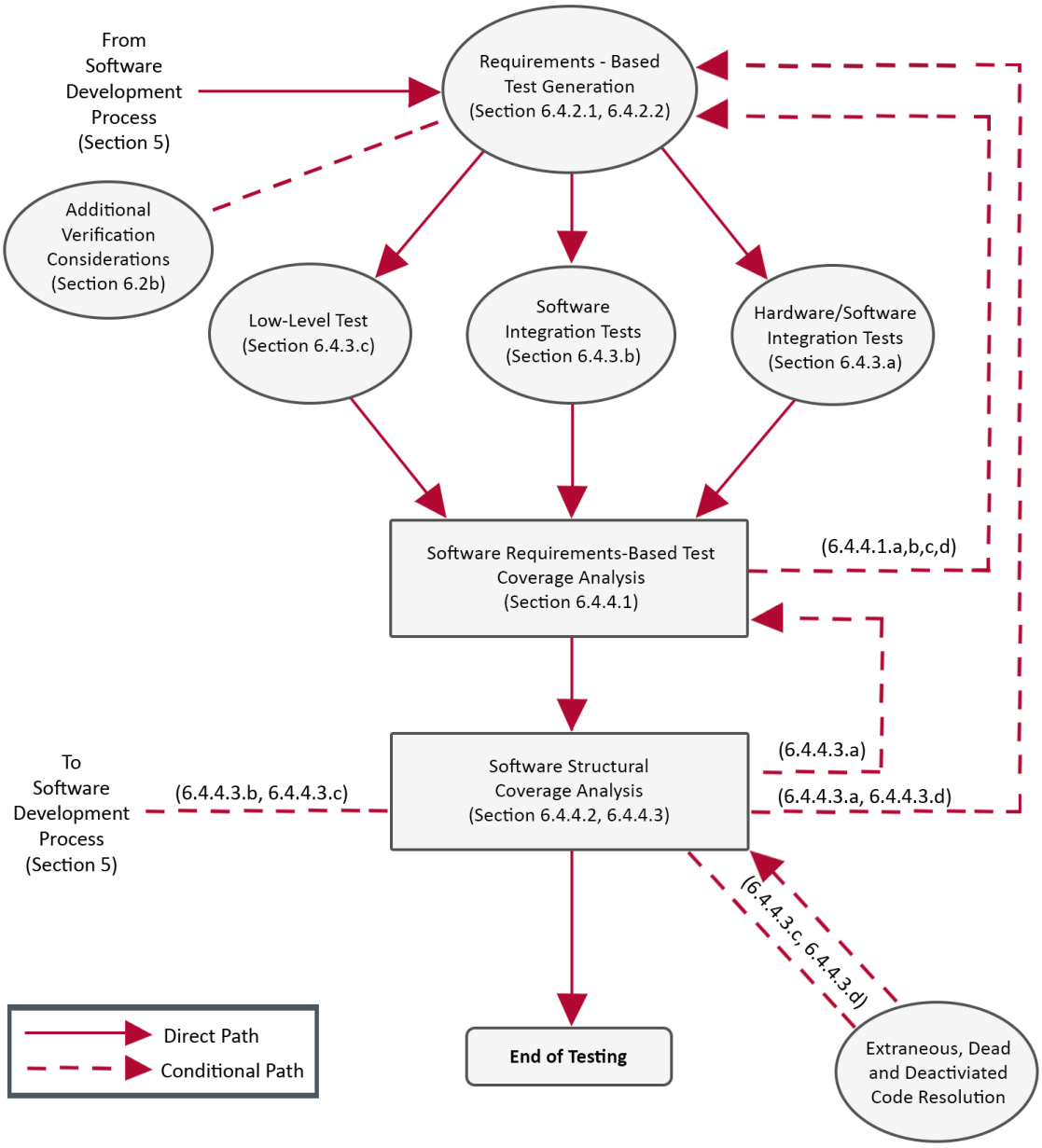
ResearchGate/Wonkeun Youn, Seung Bum Hong, Kyung Ryoon Oh, Oh Sung Ahn. 2015. *Software Certification of Safety-Critical Avionic Systems: DO-178C and Its Impacts*. [ONLINE] Available at: https://www.researchgate.net/publication/276153236_Software_Certification_of_Safety-Critical_Avionic_Systems_DO-178C_and_Its_Impacts. [Accessed 21 May 2018].

RTCA. 2011. *DO-178C - Software Considerations in Airborne Systems and Equipment Certification*. [ONLINE] Available at: https://my.rtca.org/NC__Product?id=a1B36000001IcmqEAC. [Accessed 24 May 2018].

RTCA. 2011. *DO-330 - Software Tool Qualification Considerations*. [ONLINE] Available at: https://my.rtca.org/NC__Product?id=a1B36000001IcfkEAC. [Accessed 24 May 2018].

2 Requirements-Based Testing For DO-178C

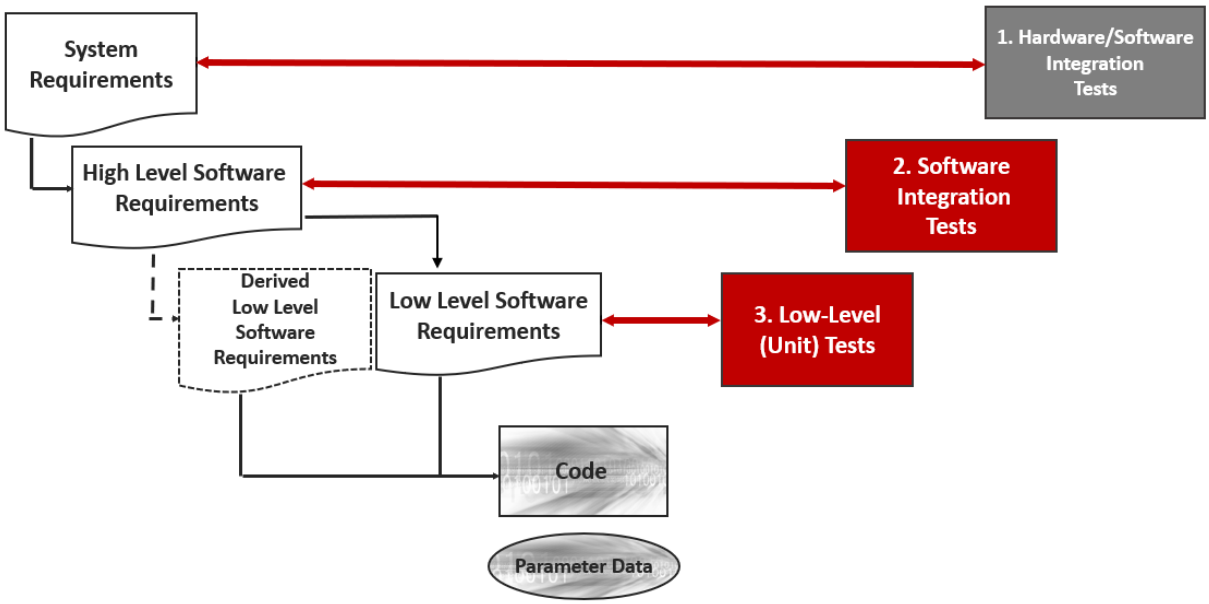
Figure 6-1 from DO-178C shows all software testing activities that are recommended. It also shows how they relate to each other as well as the sections of the standard that provide detailed guidance on them.



Three types of testing are recommended, all of which must be requirements-based:

1. Hardware/software integration testing – verifying the correct operation of the software in the target environment
2. Software integration testing – verifying the interrelationships between software requirements and components and to the implementation of the software requirements and software components within the software architecture
3. Low level testing - verifying the implementation of low-level requirements

Each of these levels of testing correspond to an equivalent level of requirements as shown in the diagram below:



Note that DO-178C section 5.0 specifies that higher level requirements may be used as derived lower level requirements, this is shown only for low level requirements in the diagram above.

The verification process provides certifiable assurance that low-level software requirements and code have been developed according to the intended functions as defined in the system requirements.

High-level software requirements are verified as compliant against the system requirements. Then low-level requirements are verified as compliant against the high-level software requirements.

Code implementation is verified against the design (low-level) software requirements.

The verification process during this phase will provide assurance that the executable object code performs the intended functions described by the reviewed and analysed high-level and low-level requirements while executing on the target or host environment.

Test cases are developed and reviewed for complete coverage (normal range and robustness) of the software high-level requirements (software levels A, B, C and D) and low-level software requirements (software levels A, B, and C).

Test procedures are developed and reviewed to correctly and completely implement the test cases in a controlled and planned test environment.

The verification process then checks that expected results from the requirements are equal to the expected results.

2.1 Requirements-Based Test Selection

In section 6.4.2 DO-178C states that requirements-based test selection should include both "*normal range test cases and robustness (abnormal range) test case*s".

Normal range test cases test that the software responds correctly to inputs and conditions in line with the expected behaviour of the code. Automated test frameworks should be capable of the following as specified in section 6.4.2.1:

- Equivalence classes and boundary values should be used to select input variables
- Testing time related functions (e.g. filters integrators & delays) in context
- Testing state transitions possible in normal operation
- For any requirements written as logic equations, variable usage and Boolean operators should be verified

Robustness testing verifies that the software responds in an appropriate way to abnormal conditions and inputs. Again, automated test tools should be able to test the following as outlined in section 6.4.2.2:

- Equivalence classes should be used to select invalid input variables
- Abnormal system initialization
- Failure modes for incoming data
- Out of range loop counts
- Exceeded time frames
- Arithmetic overflows of protection for time related functions
- State transitions not allowed by software requirements

2.2 Requirements-Based Testing Methods

DO-178C specifies methods to be used for each of the recommended types of testing. Highlighting common errors revealed for each testing type.

In this paper we will focus primarily on low-level testing which should "concentrate on demonstrating that each software component complies with its low-level requirements". Section 6.4.3 lists the following typical errors revealed by this method of testing:

- Failure to satisfy a requirement
- Incorrect loop operations
- Incorrect logic decisions
- Processing legitimate combinations of input conditions incorrectly
- Responding incorrectly to missing or corrupt input data
- Incorrect exception handling
- Incorrect computation sequence
- Inadequate algorithm accuracy or performance

2.2.1 Coverage Analysis

DO-178C section 6.4.4 recommends both requirements-based test coverage analysis and structural coverage analysis. These two very different techniques should not be confused. Requirements coverage measures how well the requirements-based testing has verified the implementation of requirements. While structural coverage measures how much of the code has been executed by tests.

2.2.2 Requirements-Based Test Coverage

DO-178C section 6.4.4.1 clarifies that activities included should include:

- Using trace data confirm the existence of test cases for each software requirement
- Confirming that test cases satisfy the criteria of normal and robustness testing
- Resolving any issues identified – often by adding or improving test cases
- Ensuring that all test cases (and test procedures) are traceable to requirements

Table 7-A in DO -178 C recommends different coverage for different software levels as shown in the table below. For Level A Projects, both high-level and low-level requirements should also be satisfied with independence.

Table Legend:

● The objective should be satisfied with independence

○ The objective should be satisfied

– Satisfaction of objective is at applicant's discretion

Level	Test coverage of requirements	
Table A7	High Level Requirements	Low Level Requirements
Level A	●	●
Level B	○	○
Level C	○	○
Level D	○	-
Level E	-	-

2.3 Structural Coverage

DO-178C recommends structural code coverage at different levels according to the software level assigned. The following techniques are recommended:

Level	Test coverage of software structure					
Table A7	100% Statement Coverage	100% Decision Coverage	100% MC/DC Coverage	100% Data Coupling Coverage	100% Control Coupling Coverage	Verification of additional code that cannot be traced to source code
Level A	●	●	●	●	●	●
Level B	●	●	-	●	●	-
Level C	○	-	-	○	○	-
Level D	-	-	-	-	-	-
Level E	-	-	-	-	-	-

It is beyond the scope of this paper to discuss these in detail but further information can be found in our Which Code Coverage Metrics to Use Whitepaper.

If the required structural coverage is not achieved it is important to analyse why. Test cases may need to be added or improved. Code may be unreachable (i.e. “dead code”), or the code that is not covered may not be needed in a certain context (i.e. “deactivated code”).

2.4 Manual Test Generation

Crafting test cases and procedures manually from requirements is labour intensive, naturally expensive and prone to human error and inconsistencies. Even when tests have been manually created for all requirements there may still be gaps in requirements coverage if each requirement is not fully satisfied. *Reviews and Analysis of Requirements* (DO-178C part 6.3) defines that requirements (high & low level) must be accurate and unambiguous as well as being logically consistent, verifiable & conform to standards (with any deviations needing to be justified).

Although DO-178C specifies that high and low-level requirements must be verifiable, this does not necessitate that low level requirements are sufficiently decomposed and well defined as to make it easy to write test cases to sufficiently satisfy them. Multiple complex test cases might be needed to test code according to requirements. These can be difficult to produce even when requirements are correct, complete, unambiguous and logically consistent. This can lead to a high reliance on looking at the structural code coverage and reverse engineering requirements to fill the gaps.

For complex code, test vectors can be very complex making it hard to manually write test cases as well as to verify that requirements are sufficiently satisfied by the test cases. It can be difficult to manually calculate which specific combinations of pre-conditions & inputs will drive the code down a path to create the correct expected behaviours/outputs, and post conditions.

2.5 Automatic Test Generation from Requirements

The feasibility of automatic test generation from both high and low-level requirements is entirely dependent on how requirements have been defined. Manually or automatically generating test cases from requirements written in natural language presents several problems. The form in which requirements are often written (e.g. use cases or functional descriptions) does not necessarily translate naturally into the individual implementation configuration items that need to be tested for low level requirements-based testing. This issue is compounded when high-level requirements are used as derived low-level requirements (specified in DO-178C section 5.0). This can be partially resolved by writing requirements in structured natural language, or programming design language, or as mathematical specifications. However, these techniques mean that requirements are more problematic to write and limit the intended functionality of what can be defined.

More recent technical advances in structured requirements definition using modelling languages (e.g. UML) rather just words, have led to adoption of Model Based Development (MBD) and Model Based Testing (MBT). Meeting the DO-178C software verification objectives (including requirements-based testing) is covered in the supplement DO-331 '*Model Based Development*' and is beyond the scope of this paper. When applicable, these MBD and MBT techniques can have very significant cost advantages for automatic test generation from requirements, particularly when the applied to design models for low-level requirements in addition to specification models for high-level requirements.

However, it is worth noting some significant limitations with the modelling approach for requirements-based testing. Firstly, it is often not possible to generate a significant proportion of code (including parameter data) due to the limitations of the modelling frameworks. This occurs mostly commonly with low-level functionality, resulting in code that needs to be crafted manually, that is then subsequently is not available as a design model for automated MBT of the low-level requirements. The second limitation, is lack of appropriate abstraction when generating both code and tests from the same model, using the same algorithm. Finally, where MBT is applied and model simulation is used to exercise the model behaviour, there can also be restrictions on the availability of exercising the tests on the code in the target environment (i.e. under normal operating conditions).

2.6 Automatic Test Generation from Source Code

An alternative approach, particularly for low level testing, is to consider the problem from another angle. Rather than trying to write tests directly from requirements, test procedures can be auto-generated using the source code. These test procedures can then be traced to requirements using clear descriptions of the test cases to ensure that the requirement is satisfied.

Automatic test generation from source code is only practical for DO-178C requirements-based testing if:

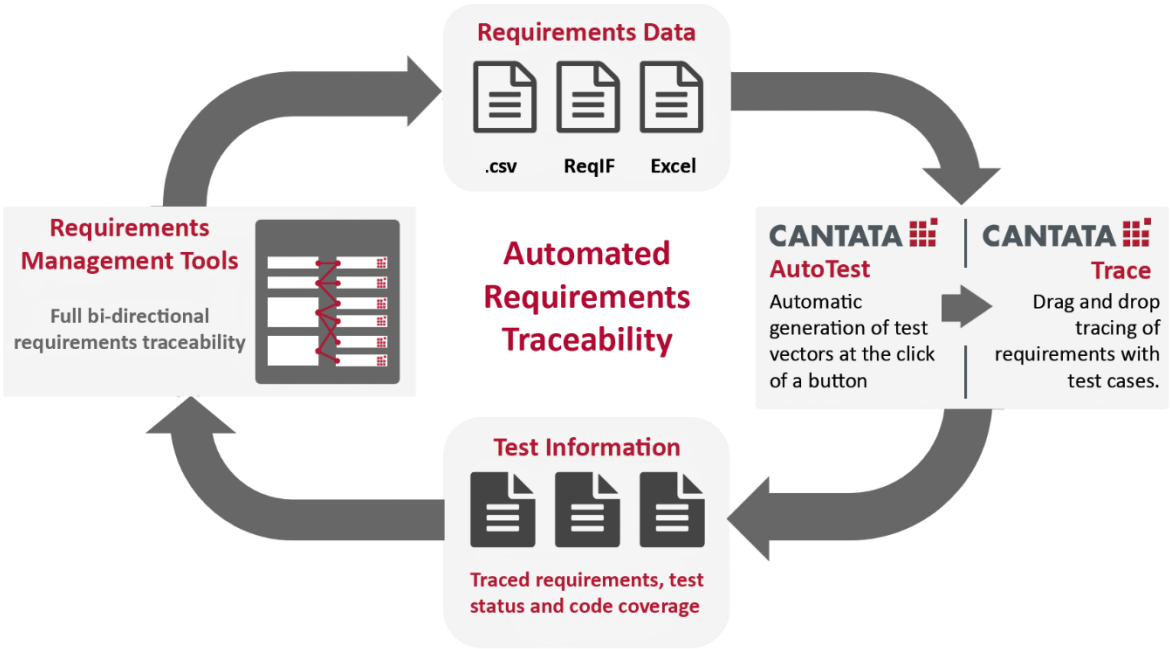
1. The process generates a near optimal number of test vectors.
2. Tests have the white box access necessary to test encapsulated code
3. Test generation is thorough – automatic test generation needs to achieve MC/DC coverage for level A projects.
4. Generated test cases and procedures are easily readable and maintainable

The validation of tests (by review and analysis) cannot be fully automated as human insight and experience are necessary to validate whether test cases satisfy the software requirements. However, test tools can provide a framework to make tracing tests to software requirements a logical and much easier task. For this it is helpful if all information required is presented clearly in a single view with natural language descriptions of the test cases and the details of the test procedures.

In the following section we will take a detailed look at how this process works for low level unit and integration tests using the Cantata tool.

3 Automating Low-Level Requirements Based Testing Using Cantata

Requirements data can be imported into Cantata and traced to auto-generated test cases. An overview of this process is shown in the diagram below:



3.1 Auto-Generating Test Cases with Cantata AutoTest

Cantata's AutoTest capability is optimized for automatic test generation for low-level requirements-based testing for C code. AutoTest parses source code to automatically generate suites of passing unit test cases for the entire code base. These test case vectors exercise all code paths, using white-box access to set data, parameters and control function call interfaces.

As well as exercising the code, tests check the parameters passed between functions, values of accessible global data, order of calls and return values. The level of detail in AutoTests can be set to meet any of the structural coverage levels specified by DO-178C (function entry point, statement, decision, MC/DC), and tests can be run on embedded target platforms as required by section 6.4.4.2

of DO-178C. AutoTest optimises path generation to create the minimum number of test cases needed to exercise all of the code according to the code coverage level required. This can result in significantly less test cases than may be produced manually in trying to satisfy the same conditions.

Tests generated by Cantata AutoTEST are easily matched with requirements (or vice versa – matching requirements to test cases) as each test case generated has a corresponding English language description of the test. AutoTEST can also substantially reduce the effort of attaining 100% requirements coverage, making it quick and easy to identify gaps in requirements, bugs in the code and unintended functionality.

3.2 An AutoTest Example

Cantata AutoTest Generation Report
Project: AutoTest Multi File Demo

Summary Information

Hostname	localhost.localdomain	Cantata	v8.0.0
Coverage RuleSet	100% Entry Point + Statement + Call + Decision Coverage	Configuration section	i686-Linux-2.6.29.4-167.fc11.i686.PAE-gcc4.4.0
Summary generated	Mar 9, 2018 10:24 AM	Time taken	2 hr 3 min

Files

Fully tested files	501	93%
Partially tested files	40	7%
Untested files	0	0%
Total	541	-

Functions

Fully tested functions	767	95%
Untested functions	40	5%
Total	807	-

In this example Cantata AutoTest was run on over 55 kloc of executable C code in 541 source files. A complete suite of Cantata in-depth isolation unit tests for 100% entry-point, statement and decision coverage were generated in just over 2 hours.

These tests were then executed using the automatic Cantata Makefile structure in just over ½ an hour.

Cantata Test Results Summary
Project: *AutoTest Multi File Demo* **Overall Result: Fail**

Summary Information

Hostname	localhost.localdomain	Cantata	v8.0
Summary generated	Mar 10, 2018 1:25 AM	Time elapsed during test run	36 minutes and 27 seconds

Build Summary

Total tests	541
Compile attempted (files)	1082
Link attempted (tests)	541
Execute attempted (tests)	541

Results Summary

PASSED	501
FAILED	40
Total checks	336355
Total checks failed	112
Total script errors/call failures	0

Coverage Summary

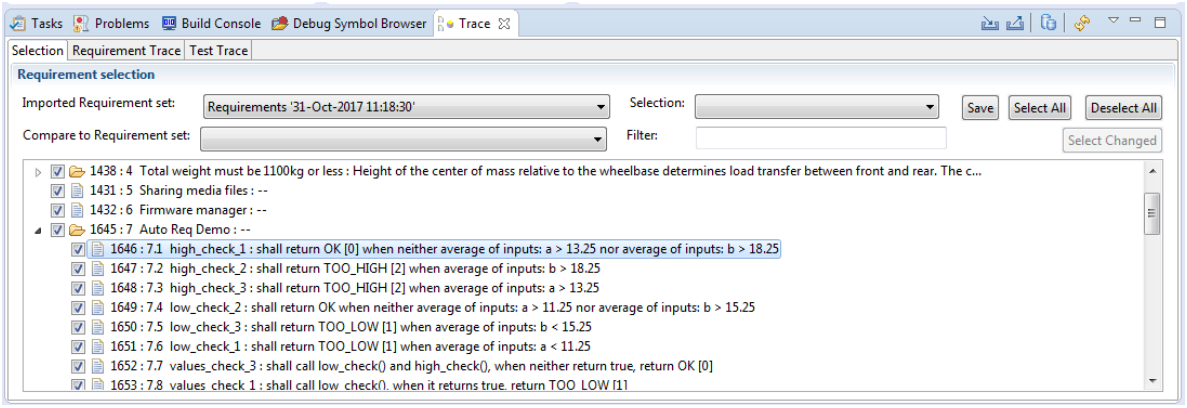
Entry point (E)	99%
Statement (S)	95%
Decision (D)	95%
Call-return (C)	95%
MC/DC - masking (M)	-
MC/DC - unique cause (U)	-

The code coverage achieved on these tested source files was incredibly high. With more than 6 dynamic checks per line of code, and a remarkably optimal set of only 5,000 test cases for over 4,900 decision outcomes (aggregate McCabe Cyclomatic complexity). The only failures were the 112 coverage failures for the 40 files where coverage targets were not met.

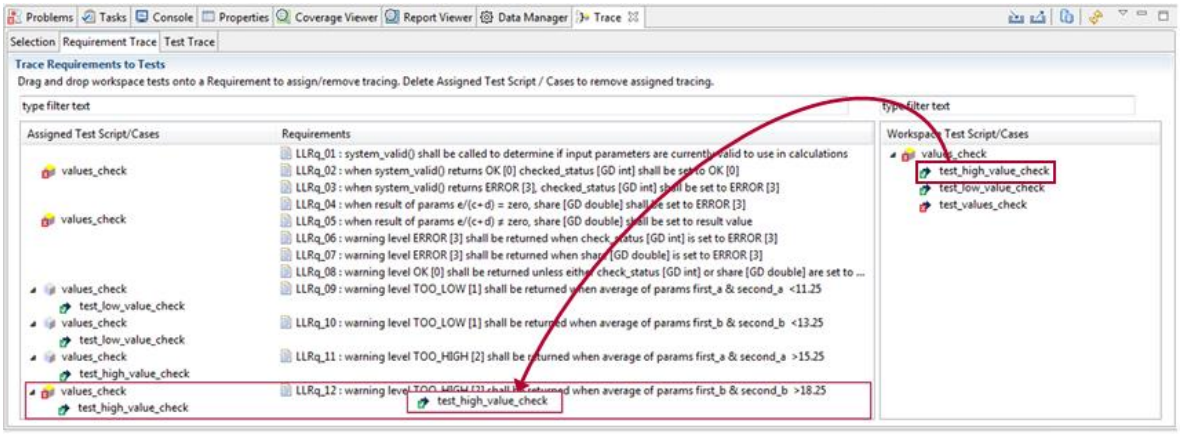
3.3 Requirements Traceability Using Cantata Trace

Requirements or test plans can be imported from any requirements management or ALM tool using ReqIF, xls, xlsx or csv formats, and are displayed in the Cantata Trace view.

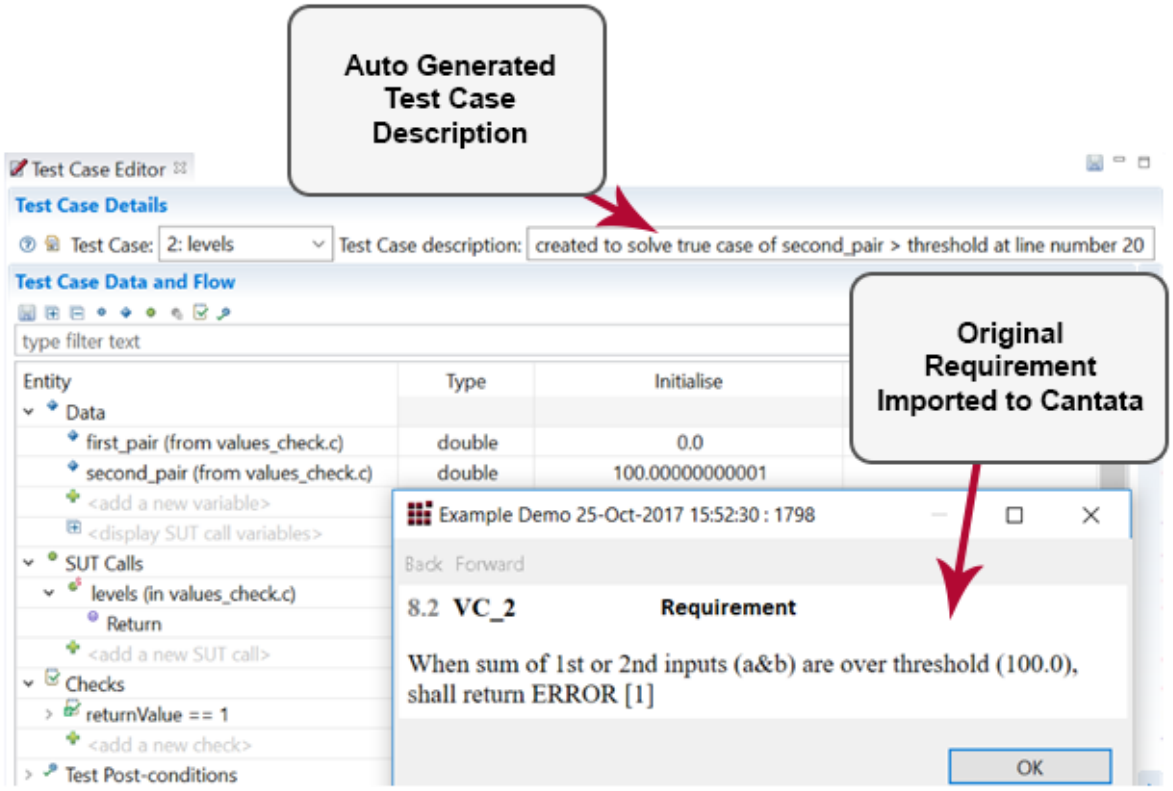
Please visit our website for a list of popular ALM tools with a Cantata integration



Drag a drop can then be used to associate test scripts/cases with imported requirements.



At this stage it is necessary to consider whether each requirement is completely and correctly verified by the test cases assigned to it. This step needs critical thinking so is not possible to automate. However, AutoTest makes this step much faster and easier, with an English description of the unique path through the code that each test case verifies. It is significantly easier to match these tests with English descriptions to the appropriate requirements than it is to split a requirement or requirements into these low level composite parts and write tests for them.



AutoTest generates passing tests from code, so if the code does not satisfy the requirements, then the tests cases should not be traced to the requirements. Once a bug is identified in this way, and the code is fixed, it is easy to update the AutoTest case so that it passes. This is usually as simple as changing a parameter or return value in the Cantata GUI test case editor.

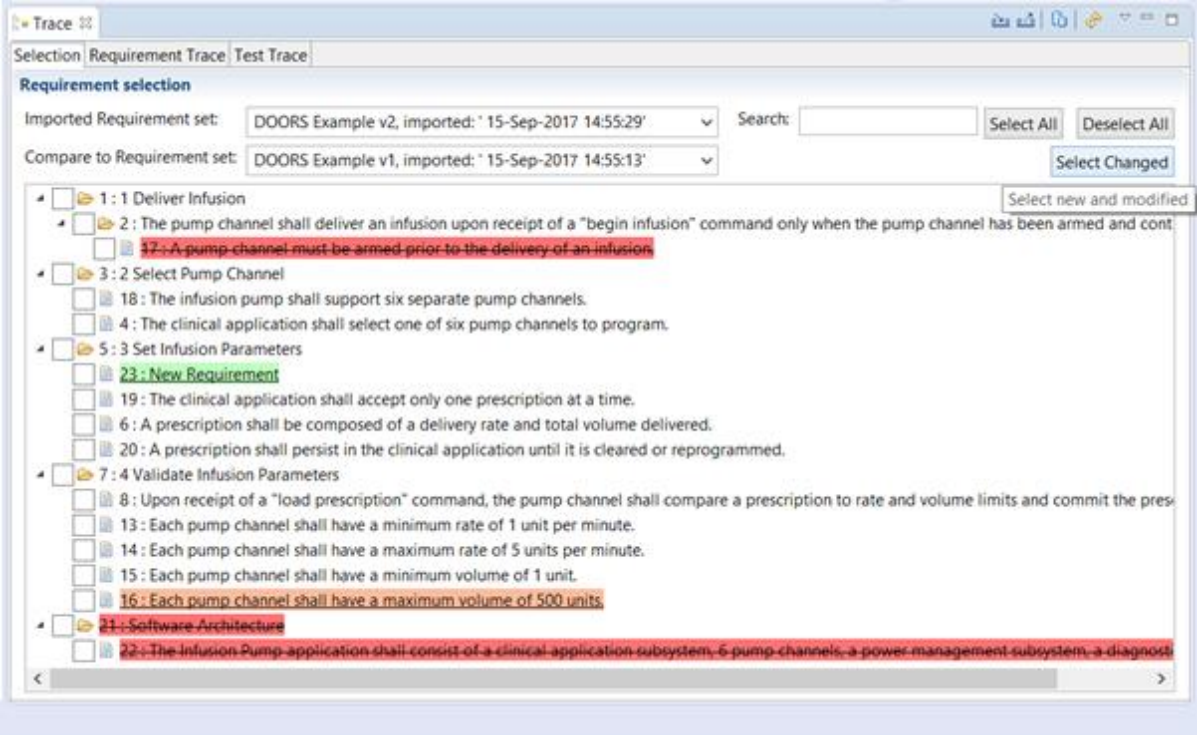
It is also easy to identify requirements that have not been implemented in the code because no tests can be traced to the un-implemented requirement. AutoTest generation is comprehensive (reporting where tests are not generated and why, so that this can be investigated), so if requirements are not fully satisfied then it is clear that they have not been implemented fully in the code.

There may also be test cases that do not match any requirement, in these cases a decision is needed as to whether the functionality of that bit of code is necessary. In which case a corresponding requirement can be written. Common causes of this would be defensive programming and system initialisation code that was not defined in the requirements. Alternatively, if the functionality should not exist (e.g. because it is “dead” or “deactivated”) then the code should be edited, and tests rerun.

3.4 Exporting and Reporting

Associations between tests and requirements are exported to the requirements management tool along with associated code coverage and test status. This exported trace data is stored in the ALM tool and useable there as trace data for DO178C certification.

As requirements change, managing the differences between versions / variants, and their relationships to existing tests can become a problem. In Cantata Trace, differences between requirement sets are highlighted, clearly showing new (green), changed (orange) and deleted (red) individual requirements. Previously traced relationships between requirements and existing tests can easily be retained for new versions and variants of requirements sets.



3.5 Automating Regression Testing

Once a correct set of passing tests has been traced to requirements they can be rerun automatically using a continuous integration tool such as Jenkins® or Bamboo® to provide an automated set of regression tests. Setting up regression tests to run every time code is built allows regression errors to be identified quickly before affecting other code items or later stages of testing. Regression testing is required by DO-178C in section 7.2.4 as part of change control. Automating this for the large number of changes throughout the development lifecycle is essential for an efficient verification process.

See our Cantata Technical Note – Integration with Jenkins for detailed instructions on setting up an integration between the Cantata unit/integration testing tool and the Jenkins CI tool.

4 Tool Qualification

Tools used for automating DO-178C verification processes (such as testing) need to be qualified for each project. RTCA DO-330 is a companion document to DO-178C, titled Tool Qualification. It describes the process and the related requirements for software tool qualification in the context of software development subject to DO-178C.

Tools are ranked by Tool Qualification Level (TQL) according to three criteria:

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) and thus could fail to detect an error, and whose output is used to justify the elimination or reduction of:

1. *Verification process(es) other than that automated by the tool, or*
2. *Development process(es) that could have an impact on the airborne software.*

Criteria 3: A tool that, within the scope of its intended use, could fail to detect an error.

4.1 Cantata Qualification

Cantata is classified as Tool Qualification Level 5 (TQL-5) according to criteria 3 above. As a COTS (commercial off the shelf) tool, the chapters *Qualifying COTS Tools* [DO-330 section 11.3] and *Tool Qualification Objectives* [DO-330 Annex A] provide useful guidance on tool qualification. Cantata has been successfully qualified for many D0-178C projects as a DO 330 TQL-5 tool.

Tool venders for COTS tools often provide evidence and guidance to aid tool qualification and suitable use of the tool. A DO178C Standard Briefing for Cantata provides detailed guidance on using Cantata to satisfy the software verification objectives of the standard.

A DO-178C Tool Qualification Kit is available for each version of Cantata containing:

- Tool Operational Requirement (TOR) Activities & Cantata Developer TOR
- Cantata operations compliance with Developer-TOR
- Cantata Configuration Management Process
- Quality Assurance Process
- Cantata Safety Manual
- DO-178C Standard Briefing paper for guidance on the use of Cantata
- Cantata Results Validity Tracker (which identifies any problems in Cantata where the results may erroneously report a successful outcome)

If you found this content useful, please feel free to share.

QA Systems
With offices in Waiblingen, Germany | Bath, UK | Boston, USA | Paris, France | Milan, Italy
www.qa-systems.com | www.qa-systems.de