White Paper

Automating Requirements-Based Testing for ISO 26262

This paper examines how automatic test case generation can deliver significant cost savings, while satisfying the software verification recommendations of ISO 26262 ASIL A to D. We also explore what ISO 26262:2018 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

The ISO 26262 safety standard, titled *Road vehicles - Functional Safety*, is a risk-based safety standard that defines functional safety for automotive electronic and electrical safety-related systems for road vehicles. The standard was developed by adapting IEC 61508 standard *(Functional Safety of Electrical / Electronic / Programmable Electronic Safety-related systems)*, specifically for development of automotive systems. ISO 26262 is applicable throughout the lifecycle of all automotive safety-related electrical, electronic and software systems.

Two editions of the standard have been released, the second edition ISO 26262:2018, supersedes the 2011 edition for all new projects. Information provided in this paper pertains to the 2018 version of the standard, but the techniques described are also applicable for projects using ISO 26262:2011.

The verification activities outlined in ISO 26262:2018 Part 6.0, *Product development at the software level,* are labour intensive and often add significant time and 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 a significant reduction in time and cost when satisfying the software verification recommendations of ISO 26262 ASILs (Automotive Safety Integrity Levels) A to D. We will explore what ISO 26262 recommends about requirements-based testing and the practicalities of automation. While other testing methods for software verification (such as Fault Injection, Interface testing, Resource usage tests, and verification of the control flow and data flow) are also recommended by ISO 26262, they are not specifically addressed in this paper. Finally, we will take a detailed look at combining the power of an automated testing framework with techniques for efficient requirements management.

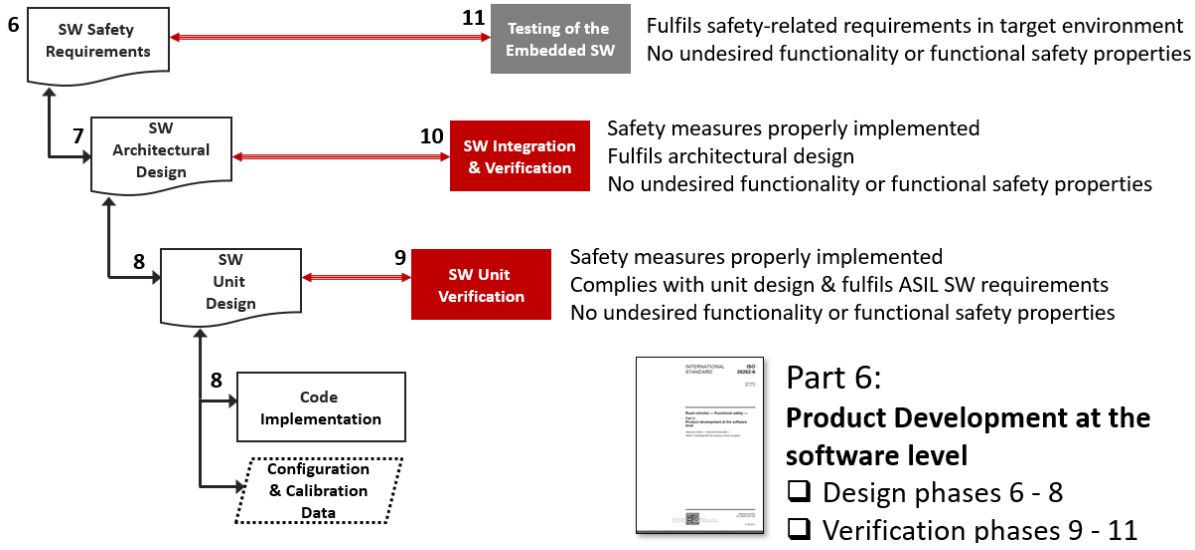
The focus for this paper is primarily on software unit and integration testing phases, because these have the most scope for automation in achieving compliance with the verification recommendations in the standard.

2 ISO 26262 Requirements-Based Testing

Requirements-based testing is integral to the varication activities required for compliance with the ISO 26262 standard. "*The objective of verification is to ensure that the work products comply with their requirements"* (ISO 26262 Part 8 clause 9.1) "*In the test phases, verification is the evaluation of the work products, items and elements within a test environment to ensure that they comply with their requirements. The tests are planned, specified, executed, evaluated and documented in a systematic manner"* (ISO 26262- 8 9.1)

Requirements-based testing is highly recommended at all ASILs for unit and integration levels of testing (ISO 26262-6 Tables 7 & 10). Requirements-based tests are also highly recommended for all ASILs for embedded software testing (Table 14).

Section 6 of ISO 26262 summarizes the phases of product development at the software level in a V-model:



The arrows in this diagram can be considered to represent traceability between phases. Verification phases on the right are traceable to the design phases on the left. Similarly, more detailed design requirements (software unit designs) need to be traceable to the higher-level requirements which they fulfil (system and software architectural design specifications, hardware-software interface specifications and software safety requirements specifications). From a practical requirements management point of view, this translates to documenting bi-directional connections between tests and requirements at appropriate levels, as well as between lower-level and higher-level requirements. Throughout the remainder of this paper, for brevity these various levels of functional and safety requirements specifications are referred to generically as just 'requirements.'

2.1 Structural and Requirements Coverage Analysis

ISO 26262- 6 9.4.4 specifies use of both requirements-based test coverage analysis and structural coverage analysis. These two forms of coverage are related but distinct. Structural coverage, often just referred to as “code coverage”, measures how much of the code has been executed by tests according to a given metric (e.g. statement coverage, branch coverage, modified condition / decision coverage).

Requirements coverage in the context of verification measures the degree to which requirements-based testing has verified the implementation of the requirements. Its purpose is to demonstrate that test cases fully verify the requirements with which they are associated.

ISO 26262 recommends using structural coverage analysis to help in achieving full requirements coverage. Considering structural coverage in terms of requirements can be a quick way to identify gaps in requirements-based tests. This technique is also useful to uncover inadequacies in requirements, dead or deactivated code and unintended functionality.

2.2 Manual Test Generation

The following methods are recommended by ISO 26262 for deriving test cases at both a unit test level (Part 6 Table 8) and for software integration testing (Part 6 Table 11):

1a Analysis of requirements

1b Generation and analysis of equivalence classes

1c Analysis of boundary values

1d Error guessing based on knowledge or experience

The first of these is obviously necessary for requirements-based testing, but consideration of requirements can also be helpful for determining equivalence classes and boundary value tests.

Crafting test cases manually using these methods is labour intensive (irrespective of test authoring framework used), naturally expensive and prone to human error and inconsistencies.

Even when tests have been manually created for all requirements, there are often gaps in requirements coverage if each requirement is not fully satisfied. *Verification - general* (ISO 26262-8 9.2b) defines that requirements must be correct, complete and consistent. If unit design requirements are atomic (so cannot be further decomposed at a unit test level), then they can be very useful for systematically producing test cases to test their successful implementation. However, although ISO 26262 specifies that all safety requirements must be verifiable (section 8, 6.4.2), this does not necessitate that lower level requirements (such as software unit design and software architectural design specifications) are sufficiently decomposed and well defined as to make it easy to write test cases to sufficiently satisfy them.

Many separate complex test cases might be needed to test code even when atomic unit level requirements are available. These can be difficult to produce even when requirements are correct,

complete, unambiguous and logically consistent. This often leads to a high reliance on looking at the structural code coverage and reverse engineering requirements to fill the gaps in tests.

For complicated code, test vectors are often 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 the necessary combinations of pre-conditions & inputs to drive the code down a path and produce the correct expected behaviours / outputs, and post conditions.

It is then perhaps unsurprising that methods for automating requirements-based testing is one of the most requested testing solutions.

2.3 Automatic Test Generation from Requirements

The feasibility of automatic test generation from requirements allocated to software architectural designs, hardware-software interface specifications, software safety requirements specifications and unit design specifications, is entirely dependent on how all these levels and types of requirements have been defined. Writing software unit design in natural language is highly recommended for all ASILS (ISO 26262-6 Table 5).

However, manually or automatically generating test cases from requirements written using natural language presents several problems.

The form in which requirements are often written (e.g. use cases or functional descriptions) does not necessarily translate easily to the individual code implementation elements to be tested during software unit or software integration phase requirements-based testing. 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 can sometimes limit the intended functionality that can be defined. ISO 26262 recommends formal notations, though does not highly recommend them for any ASILs.

Technical advances in structured requirements definition using modelling languages (e.g. UML) rather than just words, have led to adoption of Model Based Development (MBD) and Model Based Testing (MBT). ISO 26262 specifies that the compliance of the source code with the software unit design specification needs to be demonstrated for model-based development, but that this can be verified at the model level so long as evidence is provided to justify confidence in the code generator (Part 6 9.4.2). 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.

However, it is worth noting some 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 and that is then subsequently not available as a design model for automated MBT of the low-level requirements. A 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 (as required by ISO 26262-6 9.4.6).

2.4 Automatic Test Generation from Source Code

An alternative approach is to auto generate test cases from the source code and then trace these to specifications and requirements (software design specification, hardware-software interface specification and software safety requirements allocated to the software unit or component) using clear descriptions of the test cases to ensure that all the requirements are satisfied.

Automatic test generation from source code is only practical for ISO 26262 requirements-based testing if:

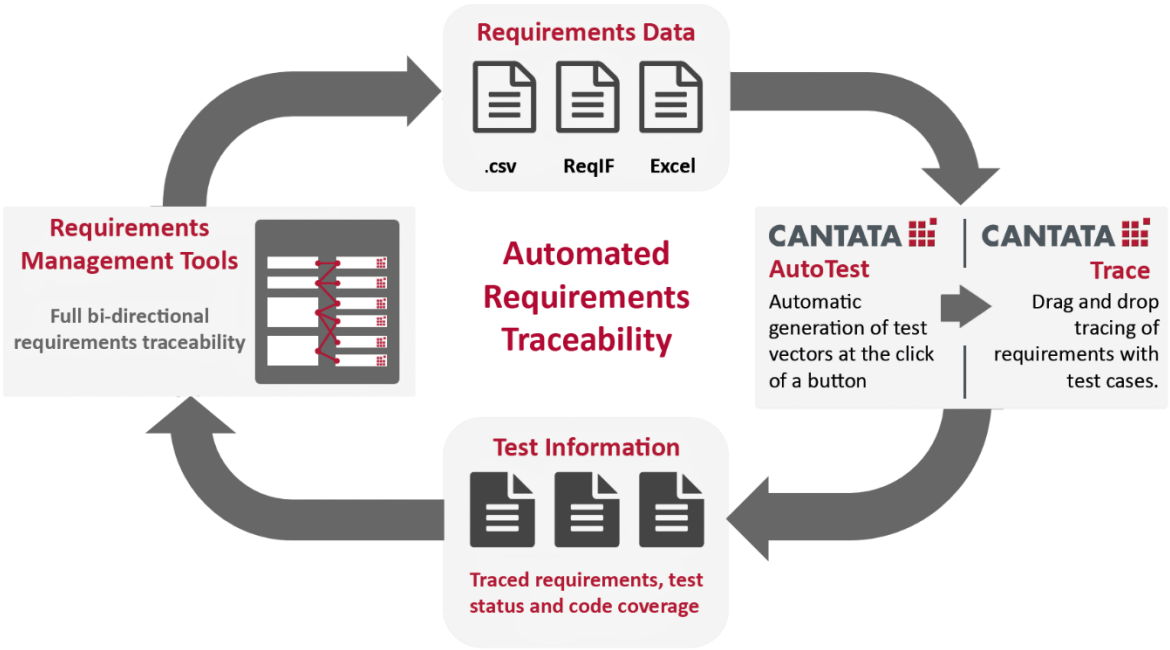
1. The process generates a near optimal number of test vectors.
2. Tests have the white box access necessary to verify encapsulated code
3. Test case generation is sufficiently thorough (e.g. achieves MC / DC structural code coverage for ASIL D projects).
4. Generated tests are easily readable and maintainable

If software requirements are the basis for requirements-based tests, any auto-generated test cases from source code need to satisfy the software requirements. To achieve this requires human insight and experience to assess the test cases. However, test tools can provide a framework to make tracing tests to software requirements a quicker and more logical task. For this it is helpful if all information required is presented clearly, with natural language descriptions of why the test cases were generated and full visibility of the test case details in a single view.

In the following section we will take a detailed look at how this process works for software unit and integration verification requirements-based tests using the Cantata tool.

3 Automating Requirements-Based Testing Using Cantata

The complete requirements data (e.g. unique identifier, textual description, tables, diagrams, hyperlinks) 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

The Cantata tool has an AutoTest capability for automatic test case generation from C and C++ source code. AutoTest can parse source code to automatically generate suites of passing Cantata test cases for each source file across the entire code base. The generated test scripts contain a set of test case vectors to exercise all code. Tests use white-box access to set inputs and check expected outputs of data & parameters as well as to control the order and behaviour of function call interfaces.

As well as exercising all paths through the code, generated tests can check the parameters passed between functions, values of accessible data, order of calls and return values. Cantata AutoTest generation preferences control the style of the Cantata test scripts and the test generation scope. Tests can be more or less thorough, testing units in isolation or as a cluster of tests within a file, and can use white-box or black-box approaches. The definition of how thoroughly generated test suites exercise paths through the code can be set to meet all the structural coverage levels specified by ISO 26262 Tables 9 and 12, and tests can be run on embedded target platforms as required by section 9.4 of ISO 26262.

The automatic generation of test case vectors from source code, setting all required input vectors (function parameters and accessible data), expected outputs (function returns and data) and expected call sequences, offers a huge productivity advantage over human testers setting up test cases – even with the assistance of the powerful test authoring capabilities available elsewhere in Cantata. The comparative advantage is even more pronounced where the complexity of the combinatorial effect of test inputs is difficult for a tester to identify.

However, for the automatic test case generation advantages to be useful for requirements-based testing, the volume of generated test cases required to exercise all the code needs to be manageable. A human tester still needs to assess each auto-generated test case to ensure it verifies the relevant software requirement. Cantata AutoTest optimises test case generation to only create a near minimum number of individual test case vectors required to exercise all the code, according to the target structural code coverage level selected.

Test cases generated by Cantata AutoTest can be easily matched with software requirements, as each test case has a corresponding English language description of the syntax for the line of source code which the test vector was generated specifically to exercise. AutoTest can therefore make it substantially easier to attain 100% requirements coverage, making it quick and easy to identify gaps in requirements, bugs in the code and unintended functionality.

More information about Cantata AutoTest is available at:
https://www.qa-systems.com/tools/cantata/autotest/

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 55,000 lines of executable C code across 807 functions 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 Cantata tests were then executed on a Windows host platform with the GCC compiler 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

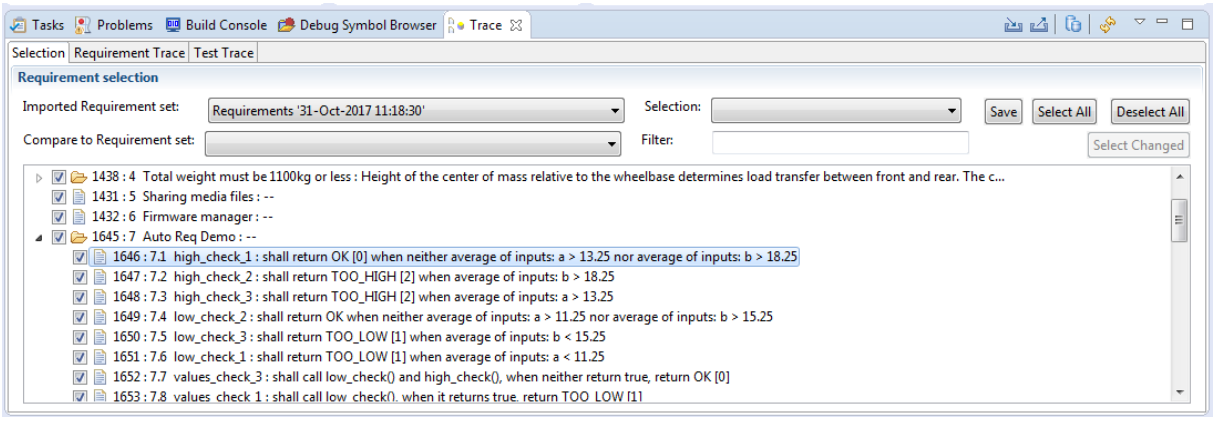
Entry point (E)	99%
Statement (S)	95%
Decision (D)	95%
Call-return (C)	95%
MC/DC - masking (M)	-
MC/DC - unique cause (U)	-

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

3.3 Requirements Traceability Using Cantata Trace

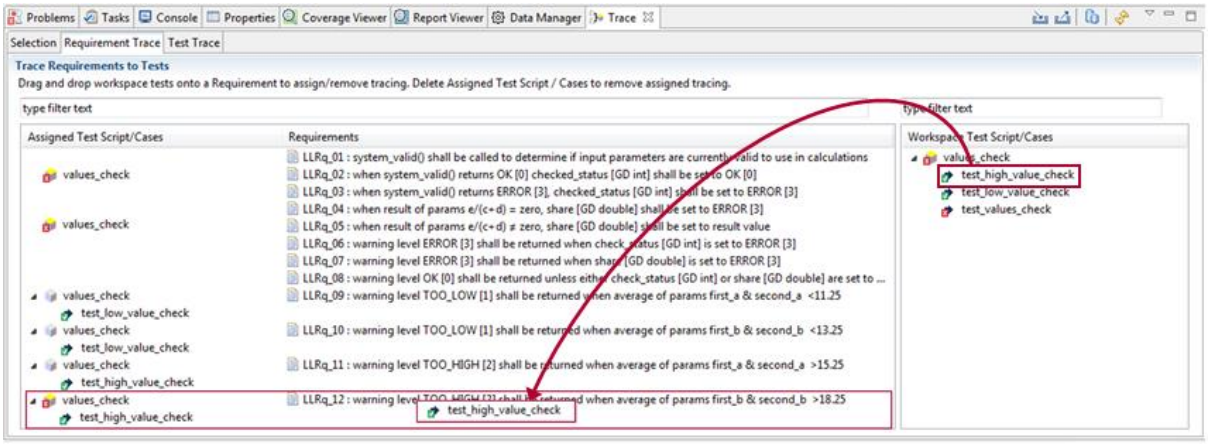
Software requirements at any level can be imported from any requirements management or ALM tool using ReqIF (Requirements Interchange Format), CSV or Microsoft Exel® formats, and are displayed directly within the Cantata Trace view.

More information about Cantata integrations with popular requirements management and ALM tools is available at: https://www.qa-systems.com/tools/cantata/requirements-traceability/



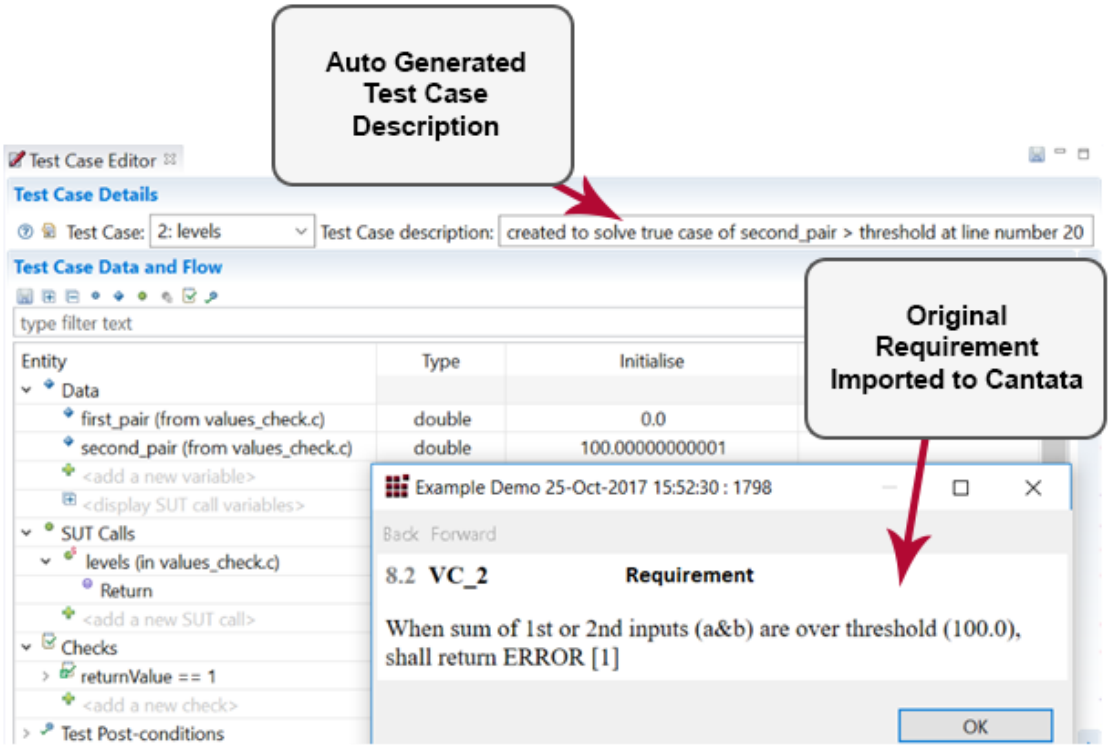
Example: ReqIF Requirements imported into Cantata Trace view for selection

The Cantata Trace view provides an intuitive drag a drop method to associate test scripts and or test case cases with imported requirements.



Example: Drag & Drop tracing a test case to imported requirements in Cantata Trace

At this stage it is necessary to consider if 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 provides an English language description of the syntax on the line of source code which the test vector was generated to uniquely exercise. These descriptions (together with the full test case details in Cantata) make it significantly easier to match the test cases to the appropriate software requirements as requirements-based tests. This tracing process is an integral part of the evaluation of tests which should be documented in the software verification specification work product (ISO 26262- 6 9.5.1) to satisfy the objective that: "*The tests are planned, specified, executed, evaluated and documented in a systematic manner*" (ISO 26262- 8 9.1)



Example: viewing requirement details alongside AutoTest test case details in Cantata

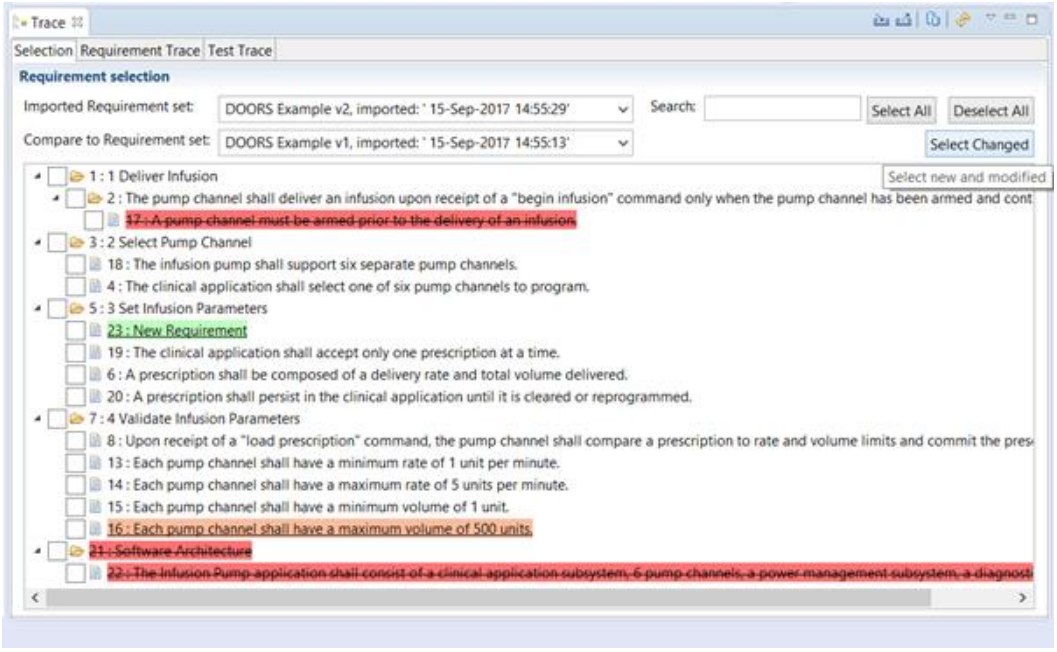
AutoTest generates passing tests from code, so if the code does not correctly satisfy the requirements, then the tests cases should not be traced to the requirements. Once a bug in code is identified in this way, and the code is fixed, it is easy to update the test case to verify the requirement so that 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 which have not been implemented in the code because no test cases will have been generated from the code that can be traced to the un-implemented requirement. The AutoTest Generation Report identifies any source files and functions where full code coverage has not been obtained 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 which do not match any requirement, in these situations a decision is needed as to whether the code implementation is necessary. If the implemented functionality is needed, the requirements may need to be updated to capture this. Common reasons this mis-match between software unit designs and code implementation include defensive programming and system initialisation code which was decomposed from, but not explicitly defined, within 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 re-run.

3.4 Managing Requirements Changes

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



Example: two versions of DOORS requirements sets compared in Cantata Trace

3.5 Exporting and Reporting

Traced associations between requirements and whole test scripts or individual test cases are exported back to the requirements management or ALM tool, along with the pass / fail status of each test case and the overall code coverage achieved by the test script. This exported trace data is stored in the requirements management or ALM tool, and may be used as part of the ISO 26262 Software verification report Work Product for the appropriate phase of verification. The format of Cantata test results and trace links to requirements varies according to the requirements management or ALM tool used.

For more information about Cantata ALM tool integrations please see: https://www.qa-systems.com/tools/cantata/requirements-traceability/

3.6 Automating Regression Testing

Once a set of passing Cantata tests has been traced to requirements, the tests can be rerun automatically under continuous integration tools such as Jenkins® or Bamboo® as 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. ISO 26262 requires a regression strategy to be in place to for each phase and subphase of the safety lifecycle with full or partial re-verification whenever changes are made (part 8 9.4.1.1 i)). Automating this for the large number of changes throughout the development lifecycle is essential for an efficient verification process.

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

4 Tool Qualification

Software tools used for automating ISO 26262 verification processes (such as testing) need to be qualified for use in development of safety critical systems. Tools are assigned a tool confidence level (TCL) based on a combination of tool impact and likelihood of error detection.

ISO 26262, Part 8 section 11 recommends independent qualification for software tools. The evaluation of the tool confidence level can be performed independently by an external party. Accredited bodies can issue a certification to confirm that the suitability of the tool for use in development up to a specific ASIL, to give complete confidence in the suitability of a tool.

4.1 Cantata Certification

The last 8 versions of Cantata have been classified with the highest possible Tool Confidence Level (TCL 1), and certified as usable in development of safety related software according to the then current standard ISO 26262:2011 up to the Automotive Safety Integrity Level (ASIL) D. Cantata was classified and certified by SGS-TÜV Saar GmbH, an independent third-party certification body for functional safety, accredited by Deutsche Akkreditierungsstelle GmbH (DAkkS). From Cantata version 9.0 onwards, Cantata will be re-certified in the same way by SGS-TÜV Saar GmbH to the now current standard ISO 26262:2018.

Free of charge Cantata tool certification kits for ISO 26262 are available to ease the path to certification. The kit contains everything needed to prove that Cantata is suitable for use in meeting the verification recommendations of ISO 26262 as well as providing comprehensive guidance to help you to achieve compliance.

The ISO 262622 Tool Certification Kit is available for each version of Cantata containing:

- Cantata Safety Manual
- Cantata SGS TUV Certificate
- Cantata SGS TUV Certification Report
- Cantata Results Validity Problem Tracker (which identifies any problems in Cantata where the results may erroneously report a successful outcome)
- Cantata Standard Briefing: ISO 26262 for guidance on the use of Cantata on ISO 26262 projects

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 08 October 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. 2018. *Cantata Standard Briefing, Issue 9 ISO 26262 Road Vehicles – Functional Safety*. [ONLINE] Available at: https://www.qa-systems.com/resources/detail/cantata-standard-briefing-iso-262622011/. [Accessed 08 October 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].

ISO. 26262:2011. *Road Vehicles – Functional Safety*. [ONLINE] Available at: https://www.iso.org/standard/43464.html. [Accessed 09 May 2012.]

ISO. 26262:2018. *Road Vehicles – Functional Safety*. [ONLINE] Available at: https://www.iso.org/standard/68383.html. [Accessed 03 January 2019.]

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