

IPL Testing Tools and DOD-STD-2167A

Executive Summary

This paper describes how AdaTEST and Cantata++ can be used within a software development to DOD-STD-2167A. In particular, it shows how AdaTEST and Cantata++ can be used to meet the verification and testing requirements of the standard.

IPL is an independent software house founded in 1979 and based in Bath. IPL was accredited to ISO9001 in 1988, and gained TickIT accreditation in 1991. Both AdaTEST and Cantata++ have been produced to these standards.

Copyright

This document is the copyright of IPL Information Processing Ltd. It may not be copied or distributed in any form, in whole or in part, without the prior written consent of IPL.

*IPL
Eveleigh House
Grove Street
Bath
BA1 5LR
UK*

*Phone: +44 (0) 1225 475000
Fax: +44 (0) 1225 444400
email: ipl@iplbath.com*



Certificate Number FM1589

*Last Update: 30/05/2002 14:39
File: 2167A.DOC*

1. Introduction

DOD-STD-2167A, 'Defense System Software Development', is one of the most widely used software development standards, being mandated by the US Department of Defense and many other defense procurement agencies. The standard is also frequently used outside of the defense industry in areas such as the aerospace and nuclear industries.

AdaTEST and Cantata++ are tools which support the dynamic testing, dynamic (coverage) analysis, and static analysis of Ada, C and C++ software. The original requirements for the tools included the suitability for testing safety critical software, although AdaTEST and Cantata++ are sufficiently flexible for the testing of any Ada, C and C++ software. This paper describes how AdaTEST and Cantata++ can be used within the DOD-STD-2167A *software development process*. It should be read in conjunction with the standard.

To assist in cross referencing to the standard, key items identified by the standard are shown in *italics* in the text of this paper.

The description consists of two parts. Section 2 gives a general description of AdaTEST and Cantata++. Section 3 reviews testing activities throughout the DOD-STD-2167A *software development process* and how AdaTEST and Cantata++ can be used in relation to the *software development process*.

2. A General Description of Adatest and Cantata++

AdaTEST and Cantata++ support the testing of Ada, C and C++ software at all levels, from testing of *Computer Software Units (CSU)*, through testing *Computer Software Components (CSC)*, testing *Computer Software Configuration Items (CSCI)*, and system testing. The facilities provided by AdaTEST and Cantata++ are summarised in Table 1.

Testing		Analysis	
Test Preparation	Dynamic Testing	Static Analysis	Dynamic Analysis
<ul style="list-style-type: none"> Scripts in Ada, C or C++ Script generation from test definition Test definition generation from CASE tool 	<ul style="list-style-type: none"> Test execution Result collection Result verification Timing analysis Stub simulation Host testing Target testing 	<ul style="list-style-type: none"> Metrics: <ul style="list-style-type: none"> Counts Complexity Flowgraphs Structure Dataflow^{*1} Instrumentation 	<ul style="list-style-type: none"> Coverage: <ul style="list-style-type: none"> Statements Decisions Conditions MCDC^{*2} Calls Call-Pairs^{*1} Exceptions^{*2} Trace Assertions Paths

Table 1 - AdaTEST and Cantata++ Facilities

*1: Cantata++ only *2: AdaTEST only

Testing with AdaTEST and Cantata++ is centred on a dynamic test harness. The test harness can be used to support testing at all levels, from *CSU testing* through *CSC integration and testing*, *CSCI testing*, and *system integration and testing*, in both host and target environments. In the host environment tests can either be run directly or under the control of a debugger. In the target environment, tests can be run directly, under an emulator, or under a debugger.

Test procedures are implemented as AdaTEST or Cantata++ test scripts. *Test procedures* for both functional testing and structural testing can be written in the respective application language, Ada, C or C++, specifying *test inputs*, *expected test results* and *evaluation criteria* for automated processing by the tool. *Test procedures* can be produced independently from the code of the application, and are fully portable between host and target environments.

Test procedures may be written directly by the user, or can be generated automatically by AdaTEST or Cantata++ from a set of *test case* definitions. *Test cases* can in turn either be written directly by the user, or can be imported from a CASE tool.

AdaTEST and Cantata++ static analysis provides static analysis of source and also instruments code in preparation for later dynamic analysis. The results of static analysis are reported in a list file and can be made available to the *test procedure* through the instrumented code. Static analysis results can also be exported to spreadsheets and databases.

AdaTEST and Cantata++ dynamic analysis provides a number of analysis commands which can be incorporated into *test procedures*. During test execution these commands are used to gather test coverage data, to trace execution, to verify data flow, and to report on and check the results of static analysis and dynamic analysis.

When a *test procedure* is executed, AdaTEST and Cantata++ produce a test report of *test results*, followed by an overall pass/fail summary at the end of the report. The *expected test results*, including the anticipated outcomes of static analysis and dynamic analysis can thus be specified in a single *test procedure* and automatically checked for compliance with *evaluation criteria*.

To assist *configuration management*, all AdaTEST and Cantata++ outputs include date and time information. The reports output by AdaTEST and Cantata++ would typically be incorporated into test logs, review logs, safety logs etc.

Instrumented code is not a compulsory part of testing with AdaTEST and Cantata++. The AdaTEST and Cantata++ test harnesses can be used in a 'non-analysis' mode with uninstrumented code.

AdaTEST and Cantata++ have been developed according to the IPL Quality Management System (QMS), which has been accredited to ISO 9001 and TickIT. The IPL QMS does not map directly onto DOD-STD-2167A, having originated from the requirements of the British Ministry of Defence, but it does fulfil all of the objectives of DOD-STD-2167A.

3. The 2167A Software Development Process

3.1. System Requirements Analysis/Design

In the *system requirements analysis* and *system design* phases the system functions are partitioned between *Computer Software Configuration Items* (CSCI) and *Hardware Configuration Items* (HWCI).

Products of these phases are:

- The *System/Segment Design Document* (SSDD);
- The *Software Development Plan* (SDP);
- A preliminary *Software Requirements Specification* (SRS) for each CSCI;
- A preliminary *Interface Requirements Specification* (IRS).

In the SRSs and IRS decisions will be made with respect to the functional and engineering requirements of the software and the external interfaces of the software.

The SDP will include statements on the *software engineering environment*, software quality, safety, risk management and testing strategy (which provides an introduction to the later *Software Test Plan*). It is at this point in the *software development process* that a decision to use AdaTEST or Cantata++ should be made.

3.2. Software Requirements Analysis

During the *software requirements analysis* phase the SRSs and the IRS are completed. The standard requires each SRS to include the definition of a complete set of *qualification requirements* for the CSCI. These *qualification requirements* can be expressed so as to take advantage of the facilities of AdaTEST or Cantata++.

At the end of the *software requirements analysis* phase, the SRSs and IRS are reviewed in the *Software Specification Review* (SSR).

3.3. Preliminary Design

In the *preliminary design* phase, the requirements for software (expressed in SRSs) and the requirements for interfaces (expressed in the IRS) are used to develop the software design and the interface design. Software design is expressed in a *Software Design Document* (SDD) for each CSCI. CSCI external interface design is expressed in the preliminary *Interface Design Document* (IDD).

The SDD will present a design for the implementation of a CSCI as a number of CSCs. Interfaces between CSCs and therefore internal to a CSCI are specified in the SDD. The SDD will also incorporate *test requirements* for each CSC, to be applied during the *CSC integration and testing* phase. DOD-STD-2167A requires CSC test requirements to include *stressing the software at the limits of its specified requirements*, such as the minimum and maximum values of data and the response times under maximum load.

The *preliminary design* phase also entails the preparation of a *Software Test Plan* for each CSCI, identifying the individual *test cases* to be applied in the *formal qualification testing* of the CSCI. *Test cases* should be phrased so as to facilitate their implementation in the *CSCI test environment*.

From an Ada point of view, a CSC can be equated to an Ada package, the interfaces of which may be specified in Ada as package specifications. The static analysis part of AdaTEST can therefore be used to help review the CSC interfaces against standards.

More details of how AdaTEST and Cantata++ can be used to support the *formal qualification testing* of a CSCI are given in the section of this paper addressing the CSCI testing phase.

At the conclusion of this phase, the *Preliminary Design Review* (PDR) is required to address the *adequacy of data recording, reduction and analysis methods* given in the STP. It is therefore necessary to state in the STP where and how AdaTEST or Cantata++ will be used to control tests, record test output and evaluate test results.

3.4. Detailed Design

The *detailed design phase* completes the software design for a CSCI by extending the SDD to decompose CSCs into CSUs, specifying design of the CSUs and fully specifying all interfaces internal to the CSCI. It is important to consider the testability of CSCs and CSUs as an integral part of the design process. The testing information in the SDD will be used to develop CSC and CSU testing:

- *Test cases and test schedules for CSC integration and testing* will be developed and recorded in a *Software Development File* (SDF) for each CSC.
- *Test requirements, test cases and test schedules for CSU testing* will be developed and recorded in an SDF for each CSU.

These *test cases* have to be designed to fulfil all of the *qualification requirements* applicable to the CSC, including requirements for test coverage, performance and adherence to standards. If the testing strategy includes isolation testing or top down testing, then the specification in the SDF should also give details of when CSUs will be stubbed and simulated.

Other products of this phase are an updated IDD, completing the design of external interfaces, and the *Software Test Document* (STD). The STD provides a full description of the *test cases for formal qualification testing* identified by the STP.

At the end of the *detailed design* phase the *Critical Design Review* (CDR) is held. Amongst other review topics, the CDR is required to address that there is *adequate detail in specifying test inputs, expected test results, and evaluation criteria*, for the testing of CSUs, CSCs and the CSCI.

If interfaces are specified in Ada, then package specifications and package bodies will be a product of this phase, with the static analysis part of AdaTEST analysis being used to assist with reviews.

3.5. Coding and CSU testing

The first activity of the *coding and CSU testing* phase is to develop *test procedures*, as full AdaTEST or Cantata++ scripts, for each CSU. DOD-STD-2167A requires a *test procedure* to be developed before the CSU is coded, to help ensure the objectivity of CSU testing. The *test procedure* becomes part of the CSU SDF. All aspects of CSU testing can be incorporated into the scripts, allowing full automation of testing for each CSU:

- Specification of *test inputs*.
- Execution of tests.
- Collection of actual *test results*.
- Specification of *expected results*.
- Specification of *evaluation criteria*.
- Specification of test completeness criteria.

Evaluation and completeness criteria are automated using the AdaTEST and Cantata++ 'check' commands. A pass result for a CSU *test procedure* will only be given by AdaTEST or Cantata++ if all evaluation and completeness criteria are met.

The static analysis facilities of AdaTEST and Cantata++ allow checks for compliance with *coding standards* to be built into the *test procedure* for each CSU, providing automation of a major aspect of source code evaluation. The checks available range from simple counts of the usage of language features, through to sophisticated complexity metrics.

When a CSU has been coded, it should first be tested with AdaTEST or Cantata++ in analysis mode, so that evaluation criteria based on static analysis results are verified, and test completeness criteria based on test coverage are verified. Once a CSU has passed all tests and the required level of test coverage has been achieved, the test script should be repeated in non-analysis mode, to verify that instrumentation inserted by AdaTEST or Cantata++ for analysis has not influenced the results of tests.

Execution of a *test procedure* as an AdaTEST or Cantata++ script creates a *test results* file. The *test results* provide a record of the execution of the testing of the CSU, including diagnostics where *evaluation criteria* have failed, giving an overall pass for the CSU only when all *evaluation criteria* and other checks have passed. The test results file for each CSU should be incorporated into the respective SDF as a permanent record of CSU testing.

In preparation for *CSC integration and testing*, *test procedures* for each CSC will be developed, using AdaTEST and Cantata++ in a similar manner to the CSU *test procedures*. Development of CSC *test procedures* may proceed in parallel with *coding and CSU testing*.

3.6. CSC Integration and Testing

The primary activity in the *CSC integration and testing* phase is the execution of CSC *test procedures*. Test execution will be a largely automated activity, in the same manner as CSU testing. The *test results* files output by AdaTEST and Cantata++ should be incorporated into the respective SDF as a permanent record of CSC testing.

As a result of CSC testing and consequent design modifications, some CSU tests may have to be repeated. The automation provided by AdaTEST and Cantata++ ensures that the effort involved in *retesting* is minor.

In some circumstances, such as where a CSC interfaces to an external device, manual input to testing may be advantageous. The test harnesses provide a facility to halt test execution pending manual input of an observed test result using a 'check observation' facility.

Concurrent with the integration and testing of CSCs, the formal *qualification testing* for a CSCI specified in the STD is developed further, with the production of detailed *set-up procedures, procedures for conducting each test, and procedures for analyzing test results*.

Once all CSCs are integrated and tested, the tests specified for the CSCI in the STD are executed. Typically this will involve a variety of methods, including the use of AdaTEST or Cantata++ as a data injection, simulation and data analysis tool. Only when all CSCI tests have been successfully executed should *formal qualification testing* commence in the *CSCI testing phase*.

Faults identified by the CSCI tests will be fixed and applicable CSU and CSC tests repeated. Again the automation provided by AdaTEST and Cantata++ ensures that the effort involved in *retesting* is minor.

3.7. CSCI Testing

The *CSCI testing* phase is the formal qualification of a CSCI. The activities involved are the formal execution of the tests specified by the STD, *Functional Configuration Audit (FCA)* and *Physical Configuration Audit (PCA)*. A record of the results of *formal qualification testing* is made in the *Software Test Report (STR)*.

There are many ways that AdaTEST and Cantata++ can be used to support *formal qualification testing*, some of which are described below.

- The analysis package can be used to measure test coverage.
- Where other CSCIs or systems have to be simulated, the test harness can be used to build a scripted simulation, providing inputs to the CSCI under test, collecting and evaluating outputs. Such simulations could be executed on the same HWCI or other HWCI.
- The entire qualification process can be specified and controlled using an independent test script, with the check observation facility being used to prompt test operators and schedule the execution of each test. Test results returned to the script can then be collated and evaluated automatically.

Multiple instances of AdaTEST and Cantata++ can be used together. For example, one instance to control the qualification process, one instance to measure test coverage, three instances to simulate three other CSCIs, and one instance to simulate an external system.

Faults identified by the *formal qualification testing* will be fixed and applicable CSU and CSC tests repeated. Prior to repeating formal qualification, the applicable tests in the STP should be repeated. Once again the automation provided by AdaTEST and Cantata++ ensures that extensive *retesting* can be conducted with minimal effort.

3.8. System Integration and Testing

System integration test plans and test descriptions will have been developed outside of the DOD-STD-2167A *software development process*. AdaTEST and Cantata++ can be used to support *system integration and testing* in a similar way to its use during *CSCI testing*.

3.9. Maintenance

Following completion of system testing, a system will enter use. Throughout the use of a system problems may be identified in the software and enhancements required, resulting in software maintenance activity.

Maintenance is not part of the DOD-STD-2167A *software development process*. However, for completeness, the use of AdaTEST and Cantata++ during the on-going maintenance of software has to be addressed.

As the software is modified, testing at all levels from *CSU testing* to *CSCI testing* and *system integration and testing* will have to be repeated. Comprehensive automation of these activities using AdaTEST or Cantata++ during the initial development will enable effective and efficient *retesting* to be conducted.

There may be a requirement to re-host the software. This could arise from increases in required capacity or functionality beyond the ability of the existing hardware to support, or from obsolescence of the existing hardware. The process of re-hosting the software will necessarily involve *retesting* of the software in the new environment. The portability of AdaTEST, Cantata++ and test scripts will greatly simplify this task.

4. Conclusion

AdaTEST and Cantata++ are well suited to the development of software to DOD-STD-2167A. This paper has shown that extensive use can be made of AdaTEST and Cantata++ to provide automation during the testing and integration phases of the *software development process*. It is believed that AdaTEST and Cantata++ are the only tools to offer this comprehensive functionality.