

# IPL Testing Tools and ESA PSS-05-0 Issue 2

## Executive Summary

This paper describes how AdaTEST and Cantata++ can be used within a software development to ESA PSS-05-0 Issue 2. 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](mailto:ipl@iplbath.com)*



Certificate Number FM1589

*Last Update: 30/05/2002 14:41  
File: PSS05.DOC*

# 1. Introduction

ESA PSS-05-0 Issue 2, describes the software engineering standards to be applied for all deliverable software implemented for the European Space Agency (ESA), either in-house or by industry.

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 a project applying the ESA PSS-05-0 Issue 2 *Software Engineering 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 all testing activities required by the ESA PSS-05-0 *Software Engineering Standard* and how AdaTEST and Cantata++ can be used in relation to this standard.

# 2. A General Description of Adatest and Cantata++

AdaTEST and Cantata++ support the testing of Ada, C and C++ software principally at the levels of *software units* and *software integration* tests. The facilities provided by AdaTEST and Cantata++ are summarised in Table 1. Some aspects of the products, principally coverage and static analysis may be applied at the levels of *system* and *acceptance* tests.

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

**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 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.

Executable *test procedures* are implemented as AdaTEST or Cantata++ test scripts. *Test procedures* for both *black box testing* and *white box testing* can be written in the respective application language, Ada, C or C++, specifying *test cases* (i.e. *test inputs*, *predicted results* and *execution conditions*) for automated processing of each *test case* by the tool. *Test procedures* can be produced independently from the code of the application (i.e. against the design), and are fully portable between host and target environments.

Executable *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 executable *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 executable *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.

Static and dynamic analysis results are extremely useful as they can be used to evaluate and report on quality factors and shown conformance to standards. Examples include the automatic gathering of metrics and checking against project specific bounds and checking for conformance with project specific coding practices.

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 *test report*. The expected test results as defined in the *test cases*, 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. ESA PSS-05-0 requires that all test results be documented. A document called the *Software Verification and Validation Plan* (SVVP) is nominated as the single repository into which all test related information should be placed, including the test results. The reports output by AdaTEST and Cantata++ would therefore either be incorporated directly into the test results section of the SVVP or referenced from it. The automated nature of AdaTEST and Cantata++ regarding test execution, test checking and report generation facilitate *regression testing* which is required to verify the effect of any changes.

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 ESA PSS-05-0 Issue 2, having originated from the

requirements of the British Ministry of Defence, but it does fulfil all of the objectives of ESA PSS-05-0 Issue 2.

### **3. The ESA PSS-05-0 Issue 2 Software Engineering Standard**

#### **3.1. Introduction**

ESA PSS-05-0 is a standard for software engineering whose definition is founded on a software-only lifecycle. It is therefore expected that ESA PSS-05-0 will fit within a larger system life cycle. Prior to the start of software development, activities will have already been performed at 'system level' as part of the system development lifecycle.

Systems Engineering activities define the overall requirements for the system. The system requirements can then be decomposed into subsystems which are further decomposed into their hardware and software components. Once the need for a software item or items has been identified, an ESA PSS-05-0 compliant lifecycle for each of the software items can begin. Within ESA PSS-05-0 a software item is referred to as a *software system*, reflecting the software-only emphasis. The term *software system* will be used throughout this paper to refer to the product of aPSS-05-0 compliant development.

At the start of an ESA PSS-05-0 lifecycle it is expected that there exists a definition of the System Requirements with some means of apportioning these amongst the various hardware and software subsystems (note a software subsystem in this sense is referred to as a *software system* in PSS-05-0 terminology). Once the need for a *software system* has been identified its development begins with the first phase of the ESA PSS-05-0 lifecycle, namely *user requirements* definition.

#### **3.2. User Requirements Definition Phase**

During the *User Requirements* (UR) phase the complete *User Requirements Document* (URD) is produced together with initial issues of the various PSS-05-0 plans. The URD defines the basis upon which the software is accepted.

The complete set of outputs from this phase are:

- The *User Requirements Document*;
- The *Software Verification and Validation Plan* comprising the *Acceptance Test Plan* and V&V procedures for the next phase;
- The *Project Management Plan* for the next phase;
- The *Configuration Management Plan* for the next phase;
- The *Quality Assurance Plan* for the next phase.

During this phase, the *Acceptance Test* plan will be defined within the SVVP. It is at this point that a decision to use an automated testing tool such as AdaTEST or Cantata++ during software system acceptance should be made.

At the end of the UR phase, the URD is reviewed in the *User Requirements Review* (UR/R).

### 3.3. Software Requirements Definition Phase

In the *Software Requirements* (SR) phase, the user requirements are analysed in order to produce a set of software requirements that are as complete, consistent and correct as possible.

The main output of the SR phase is the *Software Requirements Document* (SRD). The various plans started in the UR phase are updated to cover the activities of the next lifecycle phase. In addition, the *System Test* plan is defined in the SVVP. The purpose of the *System Test* plan is to outline an approach to demonstrating that the software will meet the software requirements.

Requirements are classified according to their nature, and sections within the SRD are structured according to the various types. An ESA PSS-05-0 SRD will therefore comprise sections covering functional requirements, interface requirements, performance requirements etc. A section is provided for the specification of verification requirements over and above those mandated by ESA PSS-05-0 itself. These verification requirements can be expressed so as to take advantage of AdaTEST or Cantata++.

There are also sections in the SRD for acceptance testing and quality requirements which can define the collection of specific metrics with boundaries on their acceptable limits. The automated metrics collection facilities of AdaTEST and Cantata++ can therefore prove invaluable for checking these requirements have been fulfilled.

The end of the SR phase is marked by a formal approval being given by the *Software Requirements Review* (SR/R) of the various phase outputs.

### 3.4. Architectural Design Phase

In the *Architectural Design* (AD) phase, the requirements for software (expressed in SRD) are used to develop a software design. The design will present an implementation of the software as a hierarchy of *components*. Interfaces between *components* and therefore internal to the *software system* are specified in the ADD.

The main output of the AD phase is the *Architectural Design Document* (ADD). The various plans that were updated in the SR phase are again updated to cover the activities of the next lifecycle phase.

In addition, the *Integration Test* plan is defined in the SVVP. The purpose of the *Integration Test* plan is to outline the approach to demonstrating that the software *components* (or *software subsystems* as they are also referred to in ESA PSS-05-0) conform to the ADD.

The integration test environment should be chosen and specified carefully in order to maximise its efficiency in detecting errors. It is therefore advantageous to state where and how AdaTEST or Cantata++ can be used to control tests, record test output and evaluate test results. The actual definition of specific *test procedures* and *test cases* is deferred until the next phase. More details of how AdaTEST and Cantata++ can be used to support the testing of software components are therefore given in the section of this paper addressing the next phase.

If external interfaces need to be defined these may be separately documented in an *Interface Control Document* (ICD).

Typically, the Hierarchical Object Oriented Design (HOOD) method will be used to perform and document the ADD. The software *components* identified in the ADD will be HOOD objects. Depending upon the size of the development, the software *components* may be HOOD terminal objects (i.e. they cannot be sensibly decomposed further into child objects), or the design may have stopped before reaching the terminal level (i.e. when a large, multi-company consortia is developing a single *software system*).

If the design has been decomposed to the HOOD terminal object level and the implementation language is Ada, the lowest level software components should equate one to one with Ada packages, in which case, packages specifications may be reviewed against any interface standards using the static analysis part of AdaTEST.

The end of the AD phase is marked by a formal approval being given by the *Architectural Design Review* (AD/R).

### 3.5. Detailed Design and Production Phase

The *Detailed Design and Production* (DD) phase completes the software design outlined in the ADD and documents, codes and tests it. It therefore covers the following activities:

- Design decomposition and documentation;
- Coding of software;
- Test development and documentation (for *unit*, *integration* and *system* testing);
- *Unit testing*;
- *Integration testing*;
- *System testing*.

The detailed design process involves successively decomposing design *components* until they can be expressed as *modules* in the selected programming language. At the same time all internal interfaces to the *software system* are specified. A *critical design review* shall be performed for each major *software component* to certify readiness for implementation.

During the DD phase, *test designs*, *test cases* and *test procedures* are documented in the SVVP. *Test designs*, *test cases* and *test procedures* must be defined for each level of testing defined in ESA PSS-05-0 i.e. for the *unit*, *integration*, *system* and *acceptance* test levels.

*Test designs* specify the details of the test approach for a software feature or combination of software features and identifies the associated tests i.e. identifies which *test cases* are required.

The *test cases* have to be designed to fulfil all of the *verification requirements* from the SRD as well as those defined in ESA PSS-05-0, 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 various test plans within the SVVP should also give details of when *modules* or *components* must be stubbed and simulated.

It is important to consider the testability of the *modules*, *components* and *software system* as an integral part of the design process. The design information will be used to develop the *test designs*, *test cases* and *test procedures*.

With AdaTEST and Cantata++ the *test procedures* may be written in the development language (Ada or C/C++) and executed, i.e. they are effectively automated, removing the need for the test personnel to carry out manual step by step testing using a symbolic debugger for example. As the tests are executed they create results files that can either be stored directly in the appropriate sections within the SVVP or referenced from them.

Other products of this phase are updated plans for the next phase.

At the end of the DD phase a *Detailed Design Review* (DD/R) is held to assess the reports of the various verification activities and decide whether to transfer the software.

If the design method is HOOD and the target language Ada then AdaTEST can again be used to assist interface checking as described in the previous section.

The various levels of testing performed during the DD phase are now considered in more detail.

### 3.5.1. **Unit testing**

*Unit tests* verify the design and implementation of all *components* from the lowest level defined in the DDD (i.e. the individual *modules*) up to the lowest level defined in the ADD. According to this definition, *unit testing* also covers the integration testing of *modules* and lower level *components* as they are combined to build higher level *components*.

ESA PSS-05-0 specifies that before a *module* can be accepted every statement shall be successfully executed at least once. In addition, the most probable paths through a module should be identified and tests designed to ensure these paths are executed.

The first activity of *Unit testing* is to develop automated *test procedures* as full AdaTEST or Cantata++ scripts for each *module*. Performing *unit testing* is then simply a matter of executing the *test procedures* in either the host or target development environment.

Although not a requirement of ESA PSS-05-0, a *test procedure* should ideally be developed from its *module* design before the *module* is coded. This helps to ensure the objectivity of *module* testing. With AdaTEST and Cantata++ the *test procedures* are coded in the development language, making their development prior to coding straightforward.

The *test procedures* are documented in the *Unit Test* section of the SVVP (SVVP/UT). All aspects of *unit testing* can be incorporated into the scripts, allowing full automation of testing for each module:

- Specification of *test inputs and execution conditions* (as defined in the *test cases*).
- Execution of tests.
- Collection of actual *test results*.
- Specification of *predicted results* (as defined in the *test cases*).
- 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 *module 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 conventions* to be built into the *test procedure* for each *module*, 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 *module* 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 *module* 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 *modules* and *components*. These results include diagnostics where evaluation criteria have failed, giving an overall pass for the *module* or *component* only when all evaluation criteria and other checks have passed. The test results file serves as a *test report* for each *module* or *component* and should be incorporated into the SVVP/UT as a permanent record of testing.

When *module* testing has been completed successfully, a similar process can be applied to integrated *modules* as they are combined to build higher level *components*.

### 3.5.2. **Integration testing**

*Integration testing* is also done in the DD phase when the major *components* are assembled to build the system. These major *components* are identified in the ADD.

ESA PSS-05-0 specifies that *integration testing* shall check that all the data exchanged across an interface agree with the data structures in the ADD. In addition integration testing shall confirm that the control flows defined in the ADD have been implemented.

In preparation for *integration testing*, *test procedures* for component interfaces will be developed, using AdaTEST and Cantata++ in a similar manner to the *unit test procedures*. Development of *integration test procedures* may proceed in parallel with coding and *unit testing*. These *test procedures* are documented in the *Integration Test* section of the SVVP (SVVP/IT).

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

As a result of *integration testing* and consequent design modifications, some lower level *component* or *module* tests may have to be repeated. The automation provided by AdaTEST and Cantata++ ensures that the effort involved in *regression testing* is minor.

In some circumstances, such as where a *component* interfaces to an external device, manual input to testing may be advantageous. AdaTEST and Cantata++ provide a facility to halt test execution pending manual input of an observed test result using a 'check observation' facility.

### 3.5.3. *System testing*

*System testing* is the process of testing an integrated *software system*. This testing can be done in the development or target environment, or a combination of the two.

ESA PSS-05-0 specifies that system testing shall verify compliance with system objectives as stated in the SRD. It is suggested that activities such as end-to-end testing, stress testing, preliminary verification against the URD, etc should be carried out.

Once all major *components* are integrated and tested, the tests specified for the *system* in the SVVP 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 *system tests* have been successfully executed should the development move into the next phase.

Faults identified by the *system tests* will be fixed and applicable *module* and *component* tests repeated. Again the automation provided by AdaTEST and Cantata++ ensures that the effort involved in *regression testing* is minor.

A record of the results is made in the *system testing* section of the SVVP (SVVP/ST).

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

- The analysis package can be used to measure test coverage.
- Where other *systems* have to be simulated, AdaTEST or Cantata++ can be used to build a scripted simulation, providing inputs to the *system* under test, collecting and evaluating outputs. Such simulations could be executed on the same platform or on a different platform.
- The entire *system testing* 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 *system testing* process, one instance to measure test coverage, three instances to simulate three other *systems*, and one instance to simulate an external *system*.

## 3.6. **Transfer Phase**

The purpose of the Transfer (TR) phase is to install the software in the operational environment and demonstrate to the initiator and users that the software has all the capabilities described in the URD

The capabilities of the system are, in the first instance demonstrated by the running of provisional *acceptance tests*.

### 3.6.1. *Acceptance testing*

By demonstrating the capabilities of the software *system* in its operational environment the acceptance tests validate the software. Acceptance tests should therefore be based on the *user requirements* as stated in the URD.

*Acceptance test procedures* are documented in the SVVP (AT). AdaTEST and Cantata++ can be used to support *acceptance testing* in a similar way to that used during *system testing*.

The test results are reported in the *acceptance test* section of the SVVP (SVVP/AT).

## 3.7. **Operations and Maintenance Phase**

The purpose of software maintenance is to ensure that the product continues to meet the real needs of the end-user.

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

AdaTEST and Cantata++ can continue to be used during the on-going *system* maintenance.

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

There may be a requirement to re-host the *software system*. 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 *regression testing* 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 ESA PSS-05-0 Issue 2. This paper has shown that extensive use can be made of AdaTEST and Cantata++ to provide automation during the testing and integration activities required by this standard. It is believed that AdaTEST and Cantata++ are the only tools to offer this comprehensive functionality.