

# IPL Testing Tools and the FDA 'General Principles of Software Validation'

## Executive Summary

This paper describes how Cantata/++ and AdaTEST can be used to assist with the verification and validation of software according to the FDA 'General Principles of Software Validation; Final Guidance for Industry and FDA Staff'. In particular it shows how the tools can be used to support the testing aspects of these guidelines. It also shows that the tools have been produced to a high quality standard such that their use for dynamic testing will not compromise the safety and integrity of the software being tested.

IPL is an independent software house founded in 1979 and based in Bath. The company has carried out work in the area of safety-related systems since 1987, and its Software Code of Practice has been approved by several major customers. IPL achieved accreditation to the ISO 9001 quality standard in 1988 and ISO 9000-3 in 1992. Both Cantata and AdaTEST 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: 05/06/2002 17:08  
File: FDA Guidelines.doc*

## 1. Introduction

This paper details how the IPL software testing tools, Cantata/++ and AdaTEST can be used to support a software development process which is aiming to follow the FDA ‘General Principles of Software Validation; Final Guidance for Industry and FDA Staff’ (issued January 2002). This document is referred to as ‘the guidelines’ in the rest of this paper.

Part 2 of the paper provides a generalised overview of the functionality of the tools. Part 3 of the paper matches the tools’ capabilities against the requirements of the guidelines. It will be seen that Cantata/++ and AdaTEST are mainly useful in supporting the activities described in the ‘Testing by Software Developer’ section (5.2.5) of the guidelines.

Finally, Part 4 gives further supporting evidence for the case on the tools’ internal integrity for safety-related software developments in general.

## 2. General Description of Cantata and AdaTEST

Cantata and AdaTEST are the generic names for the families of products developed by IPL for the testing of C/C++ and Ada software. In fact, at this point in time (May 2002) there are four products available:

- Cantata, for testing C (and some C++)
- Cantata++, for testing C++ (and can test C)
- AdaTEST, for testing Ada 83
- AdaTEST 95, for testing Ada 95 (and Ada 83)

These tools have been expressly created to assist in the Software Unit and Integration testing phases of development projects using these languages. The aim is to assist the developers to produce tests which are automated, repeatable, and can be run on any host or target platform. The precise functionality and availability of the tools will vary over time, so please refer to IPL for the current detailed status of the products.

The generic functionality of the tools can be summarised as follows:

### **Dynamic Testing capability.**

- The production of test drivers for executing the software under test through a series of planned test cases. The test drivers can in the case of Cantata and AdaTEST be generated automatically from test data supplied by the user, or in the case of AdaTEST 95 and Cantata++ created in the native language.
- The setting of automated ‘checks’ on the test case outputs, to verify that these matched the expected results. These are available for both standard and user-defined types.
- The creation/generation of simulations for external subprograms or objects, to better enable the software under test to be verified in isolation from other components in the system. Two techniques are available: ‘stubbing’, for all IPL tools, and ‘wrapping’, only for Cantata++

- The detection of expected and unexpected exception raising (does not apply to Cantata).
- Timing Analysis, to verify that software execution times match performance requirements.
- The generation of a test result summary which states whether the overall test passed or failed.
- The generation of full test results output which includes diagnostic information on failed checks.

### **Test Coverage Analysis.**

- The measurement of the effectiveness of the test cases by the computation of their coverage of elements of the software under test. These elements can include any or all of the following (slightly product dependent, refer to IPL information for details): entry points, statements, decision/branches, conditions, exceptions, data values ('always' and 'at least once').
- The potential to cause an overall test failure if any coverage achieved does not reach a preset minimum level.
- The generation of coverage statistic reports, which provide execution profiles including the identification of unexecuted code elements.
- The generation of trace reports to assist debugging.
- All of the above but with tests driven by means external to the tools.

### **Static Analysis.**

- The generation of source code metrics relating to the following: language construct usage, software complexity using industry-agreed definitions of the term, file and other related metrics.
- The placement of these metrics into an ASCII file suitable for code review (not Cantata++).
- The availability of these metrics for pass/fail checks during unit and regression testing activities (not Cantata++).
- The availability of these metrics in comma-separated value (csv) format, suitable for use for converting into graphical or database forms by suitable general purpose tools.

### 3. Matching Tool Capabilities Against the Standard's Requirements

This part is a systematic analysis of the recommendations of the guidelines section by section, and the detailing of those parts where it is felt that Cantata/++ and AdaTEST can make a positive contribution. It is organised according to the sections of the guidelines. The references given in italics relate directly to section numbers/names and also text quoted from the guidelines.

#### *Section 3. Context for Software Validation.*

##### *3.1.2 Verification and Validation.*

The guidelines recommend testing as a verification activity, along with various static analyses, code inspections etc. The IPL tools have as their main purpose support of code testing, but can also generate many metrics which are useful in code analyses and inspection/reviews.

The guidelines state, *“a developer cannot test forever”*, so, *“software validation is a matter of developing an acceptable level of confidence...”* This is a view wholly endorsed by IPL and supported by the IPL tools, where coverage analysis is the main objective technique offered to determine when ‘enough’ testing has been done. Different levels of code coverage should be used according to the safety integrity level of the device. This is easily selectable from within the IPL tools.

##### *3.3 Software is Different from Hardware.*

The guidelines offer various comments:

- Branching is a significant source of code complexity, making it hard to determine correct execution by inspection.
  - This is true, which is why dynamic testing should always be done as well as code reviews. Complexity can be quantified using the IPL tools Static Analysis capability.
- New defects can be introduced into software by the process of updating.
  - This is why a set of repeatable (regression) test suites is needed to mitigate against this possibility. IPL tools allow tests to be easily automated for regression testing.
- Branching may hide latent defects until late in a product's life.
  - This is why code coverage should always be at least 100% branch coverage. Tests can be checked for this coverage level as a Pass criterion.
- Change control is especially important for software.
  - Change control should always include the requirement to repeat previous tests, preferably from a regression suite. IPL tools use text files for both input and results, allowing tests to be archived in all common CM systems.
- Even seemingly insignificant changes in software can have severe side effects.

- Again, further recommendation for a repeatable test suite.
- Component-based methodologies using OO development techniques offer developers many advantages.
  - Reusability of (OO) software components is dependent on the ability to re-use and repeat existing tests. It is a fundamental maxim of OO development that whenever a new class type is derived from an existing one, then both old and new functionality must be demonstrated by testing. Both Cantata++ and AdaTEST 95 enable development of reusable test classes for OO components.

## ***Section 4. Principles of Software Validation.***

### *4.2 Defect Prevention.*

The guidelines state, “**Software testing is a necessary activity... (though)... by itself not sufficient to establish confidence that the software is fit for intended use**”. The IPL tools can certainly help with the testing aspect, and with some aspects of the supporting activities.

### *4.3 Time and Effort.*

Building a case for the safety of a medical product takes time and effort. IPL says plan properly and allow time. Experiences of IPL customers have demonstrated that when thorough unit testing is introduced the total project time is reduced, due to the need for fewer iterations in the integration and system test phases.

### *4.7 Software Validation after a Change.*

This re-iterates the earlier point about even small changes having the potential for large unwanted side effects. The guidelines stress the value of an, “appropriate level of regression testing”. IPL tools allow the production of such test suites, which can be re-run on demand in an automated fashion.

### *4.9 Independence of Review.*

The guidelines recommend the use of independent persons for validation and verification. IPL endorses this view and the IPL tools make it easy to follow for testing.

### *4.10 Flexibility and Responsibility.*

The guidelines make the point that applications can be built up from many different software component types, ranging from in-house modules to off-the-shelf components. Appropriate levels of verification must be considered for all of these. The IPL tools can serve as a means of verification for all types of component.

## ***Section 5. Activities and Tasks.***

### *5.1 Software Life Cycle Activities.*

The IPL tools have a clear role to play in the Coding and Testing activities. Their use can also be significant in the Maintenance stage when code modification and retest will be the main activities.

#### *5.2.4 Construction or Coding.*

The principal high-level programming languages widely recognised for ‘high-integrity’ software developments are currently Ada and C. Testing in these languages is supported by the IPL AdaTEST and Cantata products.

Source code may be compiled (or interpreted) for use on a target hardware platform. It can also be convenient to create software first on a development platform which is distinct from the eventual target. It is a matter of engineering policy which method is adopted for a given project. Either way, the IPL tools will enable testing to be done wherever most convenient.

Coding guidelines may be adopted to help with maintainability, clarity, complexity management etc. Compliance with such coding standards should be monitored from the very start of coding. The IPL tools can generate many source code metrics to help with such reviews.

Traceability is an important tool to verify that all code is linked to both requirements and test procedures. With the IPL tools it is easy to identify tests or parts of tests relating to specific modules and requirements implementation.

#### *5.2.5 Testing by the Software Developer.*

During test planning it is advisable to details procedures and tools which will be used in the code verification activities. The IPL tools form a convenient set, and if wanted IPL can offer consultancy on suitable procedures.

IPL wholly endorses the list of limitations (para 3) about what testing can and cannot reasonably achieve.

The guidelines state (para 4) that an essential element of software test case specification is predicting the expected result(s). Use of the IPL tools emphasises the need to predict the test ‘outputs’ as expected results, and incorporate automated check functions to verify that the actual and expected results match. An overall test pass will result if all such checks pass, and a fail will result if any one check fails.

Para 5 lists a set of basic principles:

- *The expected test outcome is predefined.*

- Agreed. See above.
- *A good test case has a high probability of exposing an error.*
  - Agreed, but this is the task of the test designer.
- *A successful test is one that finds an error.*
  - Agreed, but of course the error must then be fixed and the test re-run. The aim is to have good tests which run and pass (i.e. can demonstrate all found errors have been fixed).
- *There is independence from coding.*
  - Agreed, and conveniently assisted by use of IPL tools.
- *Both application and programming expertise are used.*
  - A good tester will need to understand the context of the code, and also have knowledge of the programming language being used.
- *Testers use different tools from coders.*
  - Yes, testing is different from debugging.
- *Examining only the 'usual case' is insufficient.*
  - Yes, error-handling test cases should also be done, and code coverage and further techniques as appropriate.
- *Test reuse and independent pass/fail confirmation are very desirable.*
  - Yes, and both are well supported by the IPL tools.

Para 6 talks about different levels of testing – unit, integration and system testing. The IPL tools are most relevant for the unit and integration levels. We also know of many customers who use the IPL tools coverage analysis even at the system test level.

Para 7 talks about structural or white-box testing. Two techniques supported by IPL tools will be especially helpful for this:

- i. Simulation facilities (Stubs and Wrapping) to allow calls to external software units to be mimicked by the test. This can be very useful when testing an incompletely available unit of code.
- ii. Various techniques (e.g. 'Friends' for C++) for allowing access to internal code elements such as data or private methods.

Para 8 talks about the different types of structural coverage. Most of these are directly supported by the IPL tools:

- *Statement coverage.*
  - Supported.
- *Decision/Branch coverage.*
  - Supported.

- *Condition and Multi-Condition coverage.*
  - Supported.
- *Loop Coverage.*
  - Supported as part of Decision/Condition coverage.
- *Path Coverage.*
  - The IPL tools (except Cantata++) offer a variant called Path Checking.
- *Data Flow Coverage.*
  - Limited support is available for this as a Static Analysis activity with Cantata.

Para 9 talks about definition-based testing, also known as functional or black-box testing. These approaches are usually used at the integration test level, and are well supported by the IPL tools. The techniques mentioned in Para 10 (Normal Case, Output Forcing, Robustness, and Input Combinations) can all be used.

Para 11 talks about the use of advanced techniques such as statistical software testing (SST) for measuring the reliability of a software component. There has been at least one significant piece of academic (Bristol University, 2001) work which used Cantata to investigate this subject, see <http://www.cs.bris.ac.uk/Tools/Reports/Abstracts/2001-kuball-0.html>

Para 12 talks about the need for re-testing during software changes. The essential requirement for regression testing is that all tests which form part of a regression suite should be automated and repeatable. In particular this means being able to be run all such tests without human intervention to the point where a definitive pass/fail result is given. The IPL tools are particularly good at this.

Para 13 revisits the subject of the unit/integration/system levels of testing. IPL wholly endorses the main opinions offered:

- *Unit testing ensures that functionality not visible at systems level is examined by testing.*
  - It is much easier to do code verification of units with the code at the unit level, rather than trying to force unit tests during system verification.
- *Unit testing ensures that quality units are furnished for integration.*
  - Agreed. Integration testing should only be attempted with code units that have passed unit tests.
- *Integration level testing focuses on the transfer of data across internal and external interfaces.*
  - Yes, and well supported by IPL techniques such as Wrapping.

Para 14 talks about the need for control measures such as traceability analysis to ensure ‘intended coverage is achieved’. In our opinion the achievement of intended coverage is more a matter for having appropriate QA controls, but traceability analysis is not a bad thing either.

Para 16 contains a lot of sound advice about the manner of validating tests, running them, and managing the output of test runs. All of these activities are well supported by the IPL tools. For example, test data validation should be carried out by reading the test scripts before the tests are run; test results include an objective Pass/Fail statement; and tests can easily be re-run to ensure that error resolution steps have really worked.

Para 17 contains further advice on checking the ‘quality level’ of the software testing tools used in the code development process. This is answered more thoroughly in a following Section (4) of this paper.

#### *5.2.7 Maintenance and Software Changes.*

This section describes various tasks that need to be undertaken for a sound safety-based approach to maintaining existing code. Amongst the various recommendations are the need for archiving all test documentation, data and results. These form the basis against which new tests on modified code can be carried out. Failure to have this material available *‘can significantly increase the level of effort and expense of revalidating the software after a change is made.’*

The IPL tools are very suitable for this approach as test data is readable and reusable, though it must be stressed that it is the users’ responsibility to ensure that it is all properly version and configuration managed, with appropriate archiving procedures being followed at all times.

### ***Section 6. Validation of Automated Process Equipment and Quality System Software.***

#### *6.3 Validation of Off-The-Shelf Software and Automated Equipment.*

The guidelines recommend as a first step towards gaining reassurance that off-the-shelf software is suitable for use that device manufacturers should check out the supplier’s design and development methodologies. Where this is not possible the manufacturer will need to do his own ‘black box testing’.

The IPL tools can help with this for checking the correct working of software libraries and DLLs. These could be done as standalone tests of the external code, but it would be more common however to do some integration testing, possibly using the Wrapping technique (for C++).

## **4. Tool Integrity and Development Standards**

In *Section 2.4* of the guidelines it is stated that, *“Any software used to automate... any part of the quality system must be validated for its intended use...This requirement applies to any software used to automate device... testing...”* The same point is made in *Section 5.2.5* (*“Such tools should have a degree of quality no less than the software product they are used to develop.”*).

A number of arguments can be offered to justify the use of Cantata and AdaTEST in a safety-related software project. The first is that all IPL tools have been developed according to the IPL Quality Management System (QMS), which has been accredited to ISO 9001 and ISO 9000-3.

Since their release the Cantata and AdaTEST products have successfully been through a number of customer audits, principally for use on RTCA/DO-178B projects at Level A (Safety-Critical). The customers involved include Boeing Aircraft, BAE Systems, Rolls Royce AeroEngines, Sextant Avionique and Ultra Electronics. Reports on these audits can be inspected by arrangement with IPL. Customers are actively encouraged to conduct their own audit of the tools, both in development and under maintenance.