

Achieving Testability when Using Ada Packaging and Data Hiding Methods

Executive Summary

A common problem encountered by Ada designers is that the better a design is at hiding implementation details within layers of abstraction, the more difficult it becomes to test. This is especially true of designs which use object oriented methods such as HOOD.

A general solution to this problem is presented, based on the insertion of test points to package specifications. The technique is compatible with most testing tools and strategies. Specific examples are given for IPL's AdaTEST tool.

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. AdaTEST has 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 444888
Fax: +44 (0) 1225 444400
email ipl@iplbath.com*



Certificate Number FM1589

Last Update: 06/08/96 16:38
File: TST_PT.DOC

1. Introduction

Current software engineering philosophy dictates the structuring of software through layers of abstraction, with the implementation details of each layer hidden from the layer above. The Ada designer will follow this philosophy in a number of ways:

- (a) Declaring data of scope which is local to a package within the package body;
- (b) Declaring subprograms of scope which is local to a package within the package body;
- (c) Declaring packages of scope which is local to a package within the package body. Such packages could in turn declare further data, subprograms and packages;
- (d) Declaring items within the private part of a package specification.

Such layered structuring of Ada packages provides for well protected abstractions. The details of implementation are hidden from the user at each level of the design. Unfortunately, the tester is also a user. Hiding internal details from the tester detracts from the testability of the design. Section 2 illustrates this problem in more detail.

A technique used by some designers to make the tester's job easier is to add entry points to a package (in the package specification) purely for testability. Section 3 formalises this technique into the concept of a standard test point procedure which can be applied throughout a design.

The inclusion of a test point procedure in package specifications raises further issues. Should the test point be left in delivered code or deleted? If left in delivered code, what should happen if a coding error elsewhere calls it? These issues are addressed by section 4.

The concept of test points was originally developed for Ada 83. With the advent of Ada 95, further methods for implementing test points and providing visibility for testing are available. Section 5 discusses test points in Ada 95.

To illustrate the use of test point procedures in more detail, section 6 shows the use of formalised test point procedures in conjunction with IPL's AdaTEST tool.

2. The Problem

To illustrate the problem, let us look at a contrived package, as shown in figure 1. Package A provides two procedures to the user in its specification, procedure W and procedure X. Within the body of package A there is some data Y, and a further procedure Z. The body of package A also declares a further package B. Package B follows the same structure as package A.

The user of package A accesses the facilities of the package through just two procedures, A.W and A.X. These in turn access data A.Y, procedure A.Z, and package A.B.

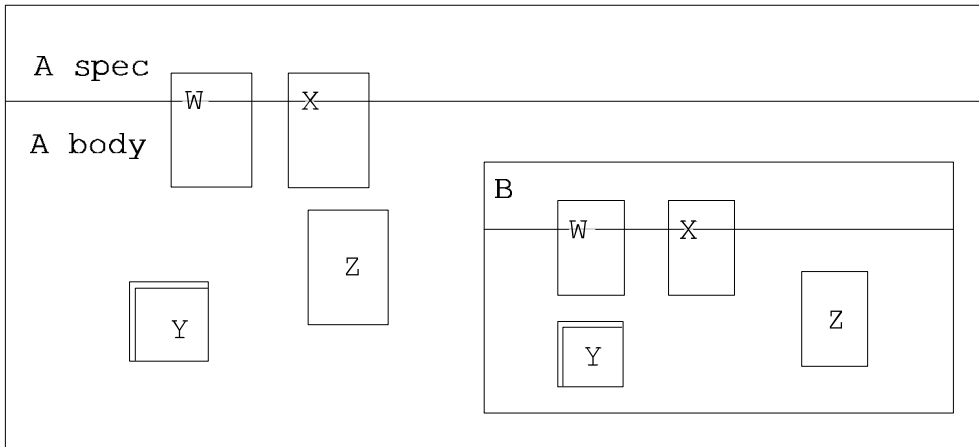


Figure 1 Items Declared in an Ada Package Body

In order to test package A fully, the tester will have to develop sequences of calls to A.W and A.X which fully test A.W, A.X, A.Y, A.Z and A.B. These sequences of calls could become very complicated, especially when testing package A.B. This can lead to a number of problems:

- (a) Excessive cost of testing due to the complexity of testing hidden components;
- (b) Incomplete testing because it is too difficult to access some components;
- (c) Inconclusive test results due to the inability to assess the internal state of the package;
- (d) A lack of helpful diagnostics when a test fails;
- (e) Situations where some hidden items are practically impossible to test;

These problems are compounded with each level of nested package arising from the layers of abstraction in a design. Overall, these problems can be classified as a lack of testability.

This testability problem may be partly avoided by declaring all packages as library items, as shown in figure 2. This gives packages a greater scope than is absolutely necessary, but avoids the complication of testing nested packages. Of course, the problem of testing the other internal components of a package are still present.

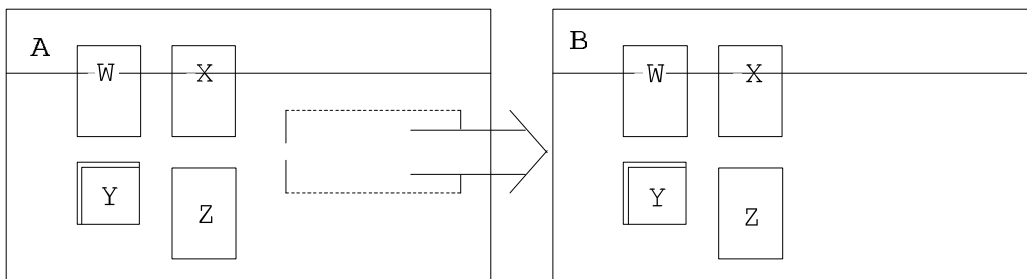


Figure 2 Declaring Packages as Library Items

An Ada designer may not wish to declare all packages at the library level. This could be due to design philosophy or due to limitations of the methodology in use. For example, HOOD designs will tend to result in a nested structure of packages as the hierarchy of objects is designed.

Clearly, a simple means of providing the tester access to the internal structure of packages is needed.

3. A Solution - Test Points

The solution proposed is the insertion of test points into the specification of packages. Such test points can then be used to provide the tester with visibility of the body of the package. This solution is not new. Many readers of this report will have used similar techniques, but maybe not in a way which is as formalised.

Taking the original example, a procedure called TEST_POINT is inserted into the specification of each package, as shown by figure 3. (Note that a test point is not needed if the package does not have a body or if nothing is declared in the package body). In figure 3, TEST_POINT is abbreviated to TP.

To avoid naming conflicts, the name TEST_POINT cannot be used for both packages A and B. The procedures TEST_POINT_A and TEST_POINT_B have full visibility of declarations inside the bodies of packages A and B. They are not constrained to the visibility of the package specification.

By declaring the procedures TEST_POINT_A and TEST_POINT_B separate from the package body, the tester can provide separate bodies for the test point procedures as part of the test harness, giving the test harness access to the bodies of packages A and B.

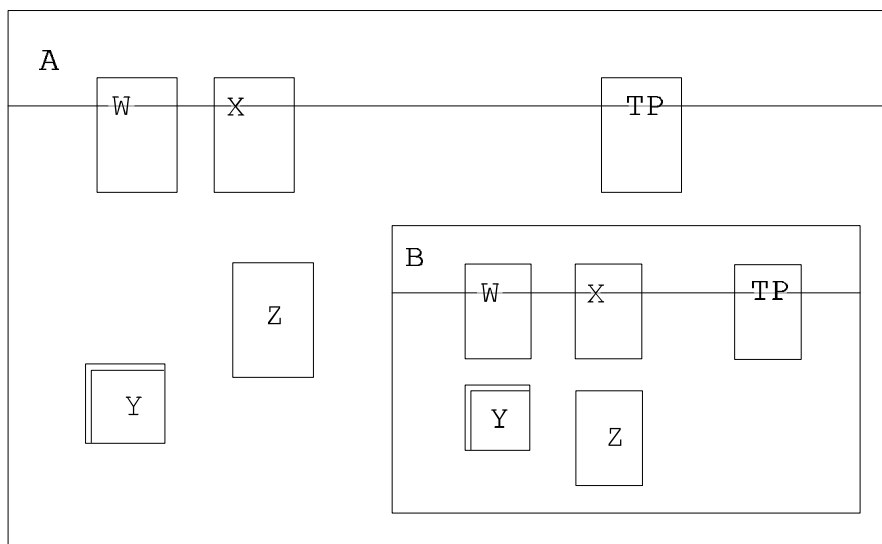


Figure 3 Test Points Inserted Into Packages

To test package A, the tester builds a test harness in two parts.

- (a) A test procedure which references package A in a "with" clause, using the visible interface provided by the package specification;

- (b) A separate body for procedure TEST_POINT_A, which has full visibility of declarations in the body of package A, including the specification of package B.

Figure 4 illustrates the structure of a test harness which uses test points. Tests are written as a cooperation between the two parts of the test harness, with the test procedure calling procedure TEST_POINT_A whenever internal access to the body of package A is required.

A test harness for package B would have to access package B through package A, using procedure TEST_POINT_A. It would therefore require an additional part to the test harness. This third part is a separate body for the procedure TEST_POINT_B.

This strategy can be applied through any number of levels of nested packages, with all parts of the test harness being coded in a single file.

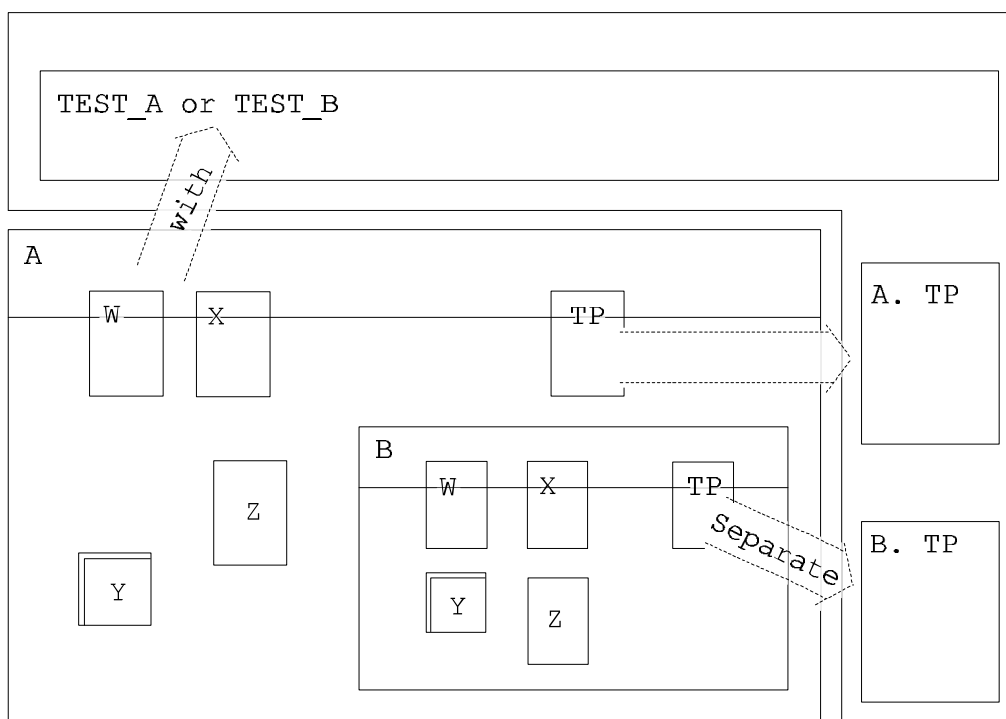


Figure 4 Test Harness Using Test Points

4. Further Issues

The previous section of this paper has described how a test point can be used to enhance testability. This solves the visibility problems encountered when testing designs which have used nested packages to model layers of abstraction. We will now look further at how test points should be used in a project environment.

The main decision to be made is whether a test point should be added temporarily for testing, or whether it should be a permanent part of a package.

If a test point is only added to a package while it is being tested, the tester will have to edit the package specification and body, insert the test point, then recompile the package. The

edit and insert operation could be automated using an edit command file. Recompilation of the package will necessitate recompilation of all dependent units.

Thus temporary insertion of test points into packages for testing can carry quite a large overhead. Another problem is that quality requirements may prohibit the modification of code for testing, because the code being tested is no longer the code which will be used. Such requirements are quite common when developing safety critical software.

The other option is to make the test point a permanent part of each package's design. This avoids any editing or recompilation of the software under test, with the test points being available to the tester from packages in the project Ada library for all levels of testing, from module testing through integration.

In the final system build, where no test points should be called, each test point should be provided with a standard body. These standard bodies are also of use for builds where a test point is not in use, such as where a package is incidental to a test rather than part of it.

Such a standard body will need to be duplicated and edited for each test point left in the final system. Its functionality could be any one of the following;

- (a) A simple "null" statement;
- (b) A statement to "raise PROGRAM_ERROR";
- (c) A statement to "raise" an application defined exception. For example a globally defined "TEST_POINT_ERROR";
- (d) Diagnostics associated with an application specific error reporting strategy.

The main point is that a test point should never be called from application code. A test point should only be called from a test harness. In the final system build there should be no calls to any of the test points in any of the packages. A good linker or global optimiser will therefore leave all of the standard test point bodies out of the build, because they are never called.

The standard test point bodies will all need testing. Effort can be minimised by producing and testing a single body. A command file can then be used to automate the production and testing of all subsequent standard test point bodies.

Alternatively, a 'test by inspection' could be justified on the basis that they should never actually be called, or even built into an executable.

If an automated design tool is being used, or a design methodology which has a very controlled relationship to the code (such as HOOD), then designing test points into the Ada as part of the normal design process could present problems. The solution is to treat test points as part of the standard structure for a package rather than as a design item.

Test points can then be incorporated when Ada designs are turned into code. If code generators are used, this can be achieved using a command file which is run after the code generator, or even as a macro in the code generator.

5. Test Points in Ada 95

Ada 95 provides the facility to extend a package by declaring a child package or child subprogram. A child provides an extension to the parent package's interface. Within a child, visibility is provided of the private part of the parent, but not of the parent's body. Furthermore, the parent has no visibility of the child. With respect to test points, a test point in the parent package cannot be used to gain visibility of the body of a child package, and a test point in a child package cannot be used to gain visibility of the body of the parent package. It is therefore necessary to provide a test point in both the parent and the child where visibility of both bodies is required.

Ada 95 also introduces the protected record, which provides synchronised access to private data within the protected record. As with packages, protected records can be declared with a test point as a protected subprogram within the protected record, giving the test point visibility of data within the protected record. However, the body of the test point cannot be declared separately, it has to be fully declared within the body of the protected record. The consequence is that functionality necessary to support testing has to be permanent part of the test point, not just an alternative body which is attached to a test script.

Test points in protected records should therefore be restricted to the minimum "utility" functionality necessary to facilitate testing:

- (a) Returning the value of data within a protected record to the test script;
- (b) (Optionally) enabling a test script to set data within the protected record.

The viability of this approach to testing protected records is consequently more dependant upon the internal design of a protected record than previous uses of test points to support testing of packages. **Protected records have to be designed for testability.**

6. Using Test Points with AdaTEST

So far this paper has discussed the use of test points in a general way which could be used in association with any test harness, including one off test harnesses. We shall now look at how test points can be used in association with AdaTEST.

The first option available is to do all of the actual testing within the test point procedure. An AdaTEST script like PDL has been used for the examples.

Example 6a

```
with A;
procedure TEST_A is
begin
  START_SCRIPT (TEST_A);
  A.TEST_POINT_A;
  END_SCRIPT;
end;

separate (A)
procedure TEST_POINT_A is
begin
  START_TEST (1);
  :
  :
  END_TEST;
```

```

:
:
START_TEST (N);
:
:
END_TEST;
end;

```

Following the opening of the test script with `START_SCRIPT`, control of all testing is passed into the test point procedure `TEST_POINT_A`. The test point procedure contains a number of test cases using the normal AdaTEST commands. If the stub simulator facility is being used, the stub simulations would follow the procedure `TEST_POINT` in the test script file.

Another option is to control all testing from the main procedure `TEST_A`. This allows some tests to act directly on the package under test through the package specification.

In the following example test 1 uses the test point to execute A.Z, but test 2 does not, executing A.W directly. Note the way in which the test number is used to control the behaviour of `TEST_POINT_A` from one test to the next.

Example 6b

```

with A;
procedure TEST_A is
begin
  START_SCRIPT (TEST_A);

  START_TEST (1);
  A.TEST_POINT_A (1);
  END_TEST;

  START_TEST (2);
  :
  EXECUTE (A.W);
  A.W;
  DONE;
  CHECK( );
  :
  END_TEST;

  START_TEST (N);
  :
  END_TEST;
END_SCRIPT;
end;

separate (A)
procedure TEST_POINT_A (TEST_NUMBER) is
begin
  case TEST_NUMBER is
  when 1 =>
    :
    EXECUTE (A.Z);
    A.Z;
    DONE;
    CHECK( );
    :
    :
  when N =>
    :
    :
  when others =>

```

```
COMMENT (Error. Test point
called when test is in
main part of script);
end case;
end;
```

This option is more complicated, but does allow greater flexibility in the control of tests. All other AdaTEST facilities can still be used in the same way as before.

When further test points are in use, such as TEST_POINT_B, it is a simple extension of the above techniques to call TEST_POINT_B from TEST_POINT_A. There are countless variations in the way in which AdaTEST can use test points.

A project should adopt a standardised approach which is compatible with the structuring of its design and the project's testing philosophy.

7. Conclusion

This paper has shown how test points can be inserted into Ada package specifications to improve the white box testability of well abstracted designs. The conflict between design abstraction and testability is removed and need no longer compromise the design.

Test points can be either a temporary addition to packages, or can be part of the permanent structure of packages. It is recommended that most projects will be best served by permanent test points. Whatever a project chooses to do, the project must be consistent.

Although Ada95 provides the means to extend packages with child packages, the principles of test points remains the same. the parent package will keep its test point, with a further test point in the child package to enhance testability of the child.

Use of test point procedures can easily be incorporated into AdaTEST scripts using a variety of techniques. The technique most applicable to a project depends on design method and test strategy. A project should be consistent in the technique it uses to incorporate test points into AdaTEST scripts.