Techniques for Testing Ada 95

S. R. Waterman
IPL, Eveleigh House,
Grove Street, Bath, BA1 5LR, UK
simonrw@iplbath.com

10th January 1997

Abstract

The Ada language is widely accepted as the language of choice for the implementation of safety related systems, and as a result much effort has been put into the identification of successful techniques for its testing. In this paper we discuss the impact of the new Ada standard upon the testability of safety related systems, and describe techniques which can be utilised to improve the likelihood of achieving testing success.

Keywords: Ada 95, Testing, Controlled Types, Protected Objects, Hierarchical Libraries

1. Introduction

The Ada language is often chosen for the implementation of safety related systems because it encourages good software engineering practices. Unfortunately, some of these practices make efficient testing difficult. As a consequence many techniques have been developed which, when applied at the design stage, can improve the testability of Ada software. These techniques remain useful when testing Ada 95 due to the similarity between Ada 95 and the previous Ada standard. In this paper we discuss the impact of Ada 95 upon the testability of safety related systems, and describe techniques which can be utilised to improve the likelihood of achieving testing success. In particular we concentrate on unit and small scale integration testing, as these are the areas likely to be affected by use of the new language features. Throughout this paper our aim is to emphasise design for testability as the primary means of achieving testing success.

In this paper we focus our attention upon three new features of the Ada language: Protected Objects, Hierarchical Libraries and Controlled Types. We begin by considering the effect that the hierarchical library has on the testing process and find that it provides the single most significant increase in testability of all Ada 95 language features. We then consider protected objects because we expect them to be one of the first features of Ada 95 to be widely accepted in safety related applications. Finally we consider the significance of using controlled types and discuss the problems that they cause during testing.

2. Testing Techniques

In this section we discuss each of our chosen Ada 95 language features in turn. We begin our discussions with a brief précis of each feature which is intended as a refresher for readers who are already familiar with Ada 95. Those with no Ada 95 experience should supplement this with a good Ada 95 textbook, for example Barnes [5].
2.1 Hierarchical Libraries

There are several drawbacks to the package model of Ada 83, which become apparent when programming large or complex systems. For example it is often desirable to use multiple library packages to model a complex abstraction, either for ease of implementation, or so that different aspects of the user interface can be encapsulated separately. In Ada 83 the only way to achieve this implementation is to put the details of the abstraction into the visible part of a package, thereby breaking the encapsulation. The alternative ‘safe’ approach results in a single monolithic package which is more difficult to use and to maintain.

A more common problem, which almost all Ada engineers have encountered at some time, is the inability to extend a package without recompiling all of its clients. This results in long and unnecessary periods of recompilation.

In Ada 95 these problems have been solved by the introduction of a hierarchical structure to the package model. In this revised model, child units can be declared which have visibility of the private part of their parent package. The simple example below is based upon that in Barnes [5] and illustrates the typical use of a child unit:

```ada
package Complex is
  type Complex is private;
  function "+"(X, Y : Complex) return Complex;
  function Real(C : Complex) return Float;
  function Imaginary(C : Complex) return Float;
private
  ... -- Private type implementing Complex
end Complex;

package Complex.Polar is
  function R(C : Complex) return Float;
  function Theta(C : Complex) return Float;
end Complex.Polar;
```

In the example `Complex.Polar` is a child package of `Complex`, and as a result the operations `R` and `Theta` can be implemented efficiently using knowledge of the internal representation of the `Complex` type.

As the name implies it is not only packages which can be child units, in fact a child unit can also be a subprogram, generic or generic instantiation.

Using Hierarchical Library Units to Increase Testability

A problem which is present to some degree at all levels of testing is how to gain access to the information hidden inside a package. The traditional way of obtaining this access is to make use of the test point technique as described by Liddiard [4]. In the simplest form of this technique, a procedure, which is known as the test point, is inserted into the package under test. This procedure serves no purpose in the normal implementation of the package and is usually given a null implementation. During testing the null implementation is replaced with test code which consequently has access to the information hidden inside the package. Unfortunately this approach suffers from two drawbacks: it is intrusive, and it is difficult to use in all but the simplest of configurations.