Software Unit Testing in an Ada Environment

Glenn Warnock
PRIOR Data Sciences

Introduction:

PRIOR Data Sciences is currently developing a validation procedure for the Ada binding of the Graphical Kernel System (GKS). PRIOR is also producing its own version of GKS written in Ada. These major software engineering projects will provide an opportunity for PRIOR to demonstrate a sound approach for software testing in an Ada environment.

PRIOR's GKS/Ada validation capability will be a collection of test programs and data, and test management guidelines. These products will be used to assess the correctness, completeness, and efficiency of any GKS/Ada implementation. GKS/Ada developers will be able to obtain the validation software for their own use. PRIOR anticipates that this validation software will eventually be taken over by an independent standards body to provide objective assessments of GKS/Ada implementations, using an approach similar to the validation testing currently applied to Ada compilers. In the meantime PRIOR will, if requested, use this validation software to assess GKS/Ada products. This project will require PRIOR to offer a well organized, thorough, and practical method for high level product testing.

The second project, PRIOR's implementation of GKS using the Ada language, is a conventional software engineering task. It represents a large body of Ada code and has some interesting testing problems associated with automated testing of graphics routines. Here PRIOR's normal test practices which include automated regression testing, independent quality assurance, test configuration management, and the application of software quality metrics will be employed.

PRIOR's software testing methods emphasize quality enhancement and automated procedures. These general methods apply to software written in any programming language. Ada makes some aspects of testing easier, and introduces some new concerns. These issues are addressed below.

The Goals of Unit Testing:

The goal of a test plan is the discovery of the maximum number of errors within a reasonable cost limit. Costs may be measured in dollars or in elapsed time, and will have different limits depending on the nature of the software being tested. For example, PRIOR is aiming to be able to validate a GKS/Ada implementation within a period of less than one week. To achieve this PRIOR's GKS/Ada test

suite will have to be carefully organized so that it is both robust, and yet still easy to use.

Testing of GKS/Ada provides an excellent example for our examination of Ada unit testing. Comprehensive and sophisticated unit tests are required to test the complex functionality. The requirements are well defined by the GKS standard, while the design specifications are covered by the proposed standard Ada binding for GKS. A unit test plan should test both the GKS requirements, and the GKS/Ada binding characteristics.

Testing Techniques:

Essentially, the purpose of unit testing is to exercise the module under test to verify that it performs correctly without producing undesirable side effects. PRIOR has developed TESTWARE, a collection of tools which provide a standard methodology to exercise and validate software modules. TESTWARE is used to initialize the appropriate global data areas and call the module to be tested with the appropriate input parameters. The returned parameters and results are then verified.

The use of a tool such as TESTWARE results in a suite of test cases which has significant value for the full life of the associated software module. An additional benefit of such a methodology is the ability to measure the degree of test coverage, to track the progression of testing, and to schedule software projects with greater accuracy.

The basic component of PRIOR's TESTWARE is the test driver. The test driver provides the framework necessary to run the tests and log the results. For each test, the necessary initializations of global data and input parameters are performed by the test driver. The module under test is called, executes and returns. The test driver must verify the return parameters and validate the global data.

In the course of execution of the module, some stubs may be necessary to "feed" the module with the necessary output parameters. It is often desirable to verify that the correct stubs are called and the appropriate input parameters passed to them. For these testing activities it would be very convenient to have an Ada compilation system that treated every call to an uncompiled subprogram as a request to interact with the test operator. The Ada system should make known the parameter values passed in, and permit the operator to supply values to be returned. We are currently writing stub routines to do this, but it would be more efficient to have this done automatically. Ada compilation systems with this capability will be very useful.

Every module to be tested requires a unique test driver. Therefore, the production of the test driver must be as automated as possible. Working from a standard

template, the test developer uses standard utilities and adds specialized code to perform the necessary initializations and verifications.

The test driver is actually driven by the test data. Data is required for initializing the global data and specifying the input parameters. Stub data is comprised of stub names, expected input parameters, and the required output parameters. Additional data describes the expected output parameters and specifies expected changes to global data. The separation of data from the test program eliminates the need to recompile the software when test data must be changed. An unlimited number of test cases can be defined in a single test data file.

Standard utilities are used to provide the translation from data to test case. The greater the flexibility available in describing test data, the more powerful and easy to use will be the testing tool. The tester should be able to easily specify enumerated types, character strings, and floating and fixed point real numbers. A range or allowable delta must be available for specifying expected output values such as floating point reals.

A variety of automated test tools such as TESTWARE have been developed for languages such as Pascal, C, and FORTRAN. These often test for errors which will not occur in Ada due to the strong typing, interface checking and run time error checking. However, additional testing difficulties arise which relate specifically to the Ada language. Testing of tasking operations is necessary to identify deadlock and starvation. Procedures for testing generic packages are required. Run time performance must also be assesed.

The GKS/Ada validation suite poses some additional problems. GKS output is often of a form which is most easily validated interactively. As an example, one test case may cause a green duck to be drawn upside down in the lower left corner. An important aspect of effective testing is that the test itself should validate the results. If the test procedure simply describes the correct display the operator may not notice if the green duck actually appears in the lower right corner. It is preferable to have the test software ask: "What colour is the duck?" (Green). "Is it upside down?" (Yes). "Is it in the lover left corner?" (No). It can be seen from this example that the task of supplying effective test software is a significant one.

The overall consideration in the design of TESTWARE is that the tester have the necessary tools to easily create the appropriate environment for running the unit under test and to be able to verify its actions and results. At the same time he must not be required to provide tedious amounts of data which are not directly related to the test.

Project Management:

Often the test portion of a software project is not given the attention or importance it deserves. Testing is usually viewed as something like "the process of

demonstrating that errors are not present" when actually errors are inherent in software. When software is tested by the person or group which developed it, with this attitude, it is not surprising that many errors go undiscovered.

To be successful testing should be approached with the philosophy expressed by Glenford Meyers. "Testing is the process of executing a program with the intent of finding errors". Testing is really a destructive process. The implementation schedule should reflect this and allow the necessary time for testing and corrections. The evaluation of test effectiveness should be based on the number of errors discovered. To be most effective it is best to have an independent test team.

Significant responsibilities must rest on the test authority. Developing a unit test for every module in the system is often not appropriate so the test authority must determine which modules should be tested and in which combination and order. The selection of appropriate test cases is critical to the success of testing.

Testing can be performed in an incremental or non-incremental manner. In the non-incremental method, all modules are tested seperately, with calls to lower modules replaced by stubs. When all modules have been tested, they are integrated and tested as a system. This method allows for greater parallelism in the unit testing process.

With incremental testing, the previously tested modules are used by the module under test, when available, instead of stubs. This provides more test coverage as the earlier modules are more extensively exercised. Also, integration and interface errors are discovered earlier and are easier and less expensive to correct.

Although top down design is often the preferred method of large system design, top down implementation and testing are not always preferrable. It is difficult to use an incremental method of testing if top-down implementation is used, as it becomes increasingly more difficult to provide the necessary input parameters to drive the test cases for the lower level modules as they are added. In addition a large number of stubs are required. With bottom up incremental testing, fewer stubs are necessary and the test driver is directly calling the module under test so that it is easier to force the test conditions.

Test cases can be generated by studying the internal logic and paths of the module (white box techniques) and by studying boundary conditions and combinations of input classes (black box techniques). Automated tools can also be helpful for this.

The real effectiveness of an automated test environment will be determined by its degree of integration into the software development environment. Test modules have to be associated with the appropriate software modules in the library. Commands should be available to permit the library manager to automatically retest appropriate modules. It is very important to track errors discovered and to have the ability to generate statistics and status information concerning the test process.

Coordination of Test Development:

A number of GKS test routines have already been written by groups in Europe and in the U.S.A. . PRIOR intends to include these in its test suite, and then extend it to cover new areas. By making this activity as visible as possible we hope to avoid any duplication of effort.