Systematic Unit-Testing of Ada Programs

Joachim Wegener, Ines Fey

Daimler-Benz AG
Research and Technology
Alt-Moabit 96 a
D-10559 Berlin
Germany

Tel.: +49 (0)30 39982-232/246
FAX: +49 (0)30 39982-107
{wegener, fey}@DBresearch-berlin.de

Abstract

The systematic test is an inevitable part of the verification and validation process for software. Overall support for all testing activities is currently not available in a single Ada testing tool. Hence, a combination of powerful testing tools is necessary to provide systematic and complete test process automation for the examination of Ada programs. The classification-tree editor CTE supports the systematic design of functional test cases. The strengths of AdaTEST are the comprehensive support for test execution and coverage analysis. The combination of both tools leads to systematic and well-documented test procedures. It has already been successfully applied to several real world examples including aerospace applications.

0. Introduction

In the aerospace and defence divisions of the Daimler-Benz Group, Ada is a widely used programming language. The systems developed with Ada are often safety-critical. Apart from a thorough way of proceeding during the development of the systems, the analytical quality assurance is of great importance for the avoidance of errors during the operation of the systems. The most important analytical method for the quality assurance of software-based systems is dynamic testing since it is the only method which allows a thorough examination of the actual run-time behaviour of such systems in their real application environment. Dynamic testing typically consumes 30 % - 50 % of the overall software development effort and budget.

Investigations of various software development projects in several divisions of the Daimler-Benz Group have shown that the costs for software testing mostly arise for unit testing, integration testing, and system testing. On average, about 35 % of the total testing expenses is spent on unit testing, 28 % on integration testing and approximately 27 % on system testing. The remaining 10 % is spent on specific tests like the examination of software-hardware interfaces.

A result of these statistics is that in practice up to 17 % of the overall development cost is allotted to unit testing. This significant quota of the entire development cost increases if integration testing is done bottom up and the integration is checked by testing higher level units. In this case the overall cost for unit and integration testing increases to an amount of 31 % of the entire development cost. This illustrates the importance of unit testing.

Significant savings can be achieved, and the product quality can be improved by tools which systematize and automate the unit testing process. Therefore, many computer-aided software testing tools exist for the testing of Ada programs. Most of these provide specialist support for distinct test activities such as test execution, monitoring, and test evaluation. Overall support for all testing activities is not currently available in a single tool. Moreover, none of the tools offers methodological support for the functional test case determination. Hence, a combination of powerful testing tools is necessary to pro-

vide systematic and complete test process automation from test case design to test evaluation.

In the first chapter this paper gives an introduction to the test activities necessary for the systematic test. Furthermore, the classification-tree method and the classification-tree editor CTE which support the systematic test case determination for the functional test are explained. In the next chapter a survey of the existing tools for unit testing of Ada programs is given. It is shown what test activities are automated by the respective tools. For an integration with the CTE, AdaTEST from IPL was chosen since it produces particularly good results in the fields of test execution and monitoring. The integration of both tools is described in chapter 3. The following chapter presents the first practical experiences with this tool combination. The paper closes with a short summary of the most important results and an outlook on future work.

1. Systematic Testing and the Classification-Tree Editor CTE

The systematic test is an inevitable part of the verification and validation process for software. Testing is aimed at finding errors in the test object and giving confidence in its correct behaviour by executing the test object with selected input values. The overall testing process can be structured into the following central test activities: test case design, test data selection, expected values prediction, test case execution, monitoring, and test evaluation. In addition, accompanying activities like test organization and test documentation are of importance. This structure facilitates a systematic procedure and the definition of intermittent results [Wegener and Pitschinetz, 1995].

During test case design the input situations to be tested are defined. A test case abstracts from a concrete test datum and defines it, only in so far as it is required for the intended test. In the course of test data selection concrete input values which meet the test case conditions are chosen. Determining the anticipated results for every selected test datum constitutes expected values prediction. The test object is run with the test data and thus the actual output values are produced. The test results are then determined by comparing expected values and actual values. Additionally, monitoring can be used to obtain information about the behaviour of the test object during test execution. A common method is to instrument the program code according to a white-box test criterion.

The most important prerequisite for a thorough software test is the design of relevant test cases since they determine the kind and scope and hence the quality of the test. A test case defines a certain input situation to be tested. It comprises a set of input data from the test object's input domain. Each element of the set should, for example, lead to the execution of the same program functionality, the same program branch, or the same program state, depending on the test criteria applied. Functional testing is of particular importance since it is the only method which allows to examine appropriately whether or not all specified requirements have been implemented in the test object. The classification-tree editor CTE supports the systematic design of functional test cases. It is based on the classification-tree method.

1.1 Classification-Tree Method

The classification-tree method [Grochtmann and Grimm, 1993] is a special approach to (black-box) partition testing partly using and improving ideas from the category-partition method defined by Ostrand and Balcer [1988]. By means of the classification-tree method, the input domain of a test object is regarded under various aspects assessed as relevant for the test. For each aspect, disjoint and complete classifications are formed. Classes resulting from these classifications may be further classified - even recursively. The stepwise partition of the input domain by means of classifications is represented graphically in the form of a tree. Subsequently, test cases are formed by combining classes of different classifications. This is done by using the tree as the head of a combina-

tion table in which the test cases are marked. A major advantage of the classification-tree method is that it turns functional test case design into a process comprising several structured and systematized steps - making it easy to handle, to understand, and also to document.

The use of the classification-tree method will be explained by testing a sample Ada function: is_line_covered_by_rectangle. The function checks whether or not a line is covered by a given rectangle with its sides parallel to the axes of the coordinate system. Input parameters of the test object are two records. The first one of type line_data describes the line by the positions of its two end points, the second one of type rectangle_data describes the rectangle using the position of its upper left corner, its width and its height. If the line is covered by the rectangle, the test object should return True, otherwise False. Figure 1 illustrates the task of the test object by means of an arbitrary rectangle and several sample lines. The figure also defines regions to describe the possible positions of the line end points with respect to the rectangle.

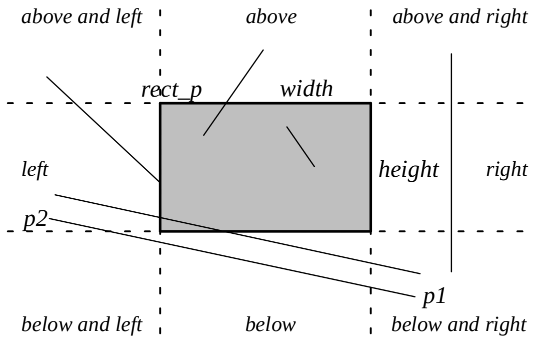


Figure 1: Rectangle with sample

The possible inputs are various lines and rectangles. Appropriate test aspects in this particular case are the positions of the line end points p1 and p2 with respect to the rectangle as well as the properties of the line like its course, orientation and gradient. Furthermore, the input domain of the test object is distinguished according to the existence and degree of coverage (Figure 2).

The classification based on the aspect degree of coverage, for instance, leads to a partition of the input domain into three classes: complete coverage (both line end points are covered by the rectangle), one end point covered (only one line end point is covered by the rectangle, the other one is located outside the rectangle), and no end point covered (both line end points are not covered, but some points in the middle of the line are covered). The classifications based on the positions of the line end points produce each a partition into two classes: line end point located outside the rectangle and line end point located inside the rectangle. Figure 2 shows a refinement for the case that p2 lies outside the rectangle. This case is further distinguished according to the position of the respective line end point with respect to the rectangle and according to the distance of p2 from the rectangle. The properties of the line are also specified in a refinement. According to the course of the line it is, first of all, distinguished between horizontal, vertical, and slanting lines. Horizontal and vertical lines are further subdivided according to their orientation, horizontal lines, for example, into lines running from left to right or from right to left. For slanting lines the gradient of the line is also taken into account. The various classifications and classes are noted as classification tree.

Afterwards, the test cases are defined by combining classes of different classifications in the combination table. Each row in the table represents a test case. For is_line_covered_by_rectangle 49 test cases were determined. The second test case, for *example, defines a test with the special case that the line end point p1 is the only point of the line which is covered by the given rectangle. P1 is located on the left side of the rectangle. P2 is located far away from the rectangle. Its position is left from and above the rectangle. A corresponding sample line is shown in Figure 1.*

A comprehensive description of the classification-tree method and related works was given by Grochtmann and Grimm [1993].

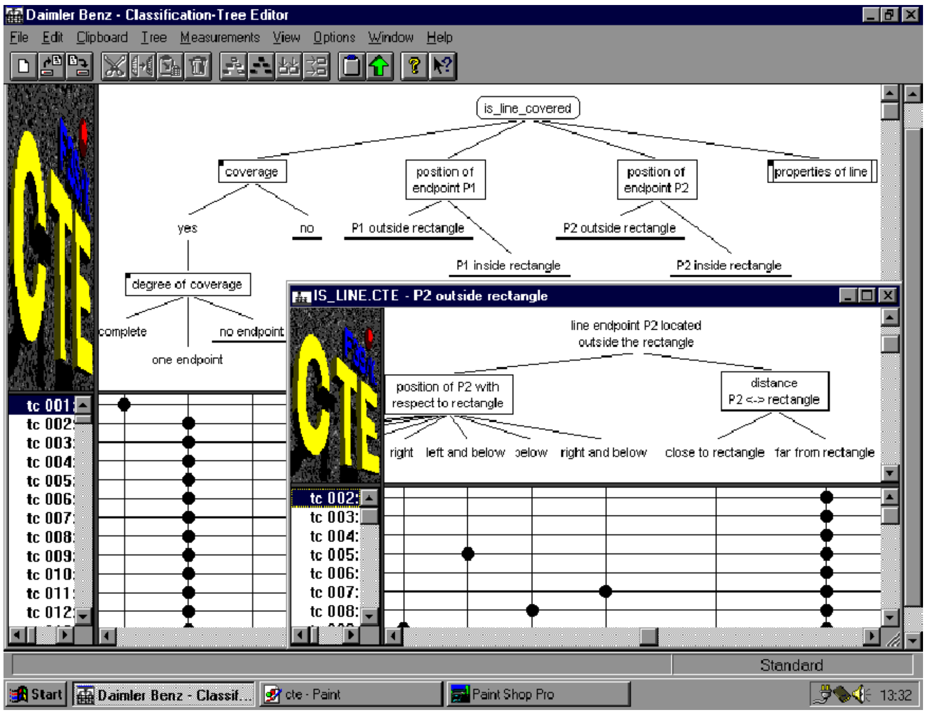


Figure 2: Test Case Design for is_line_covered_by_rectangle using the CTE

1.2 Classification-Tree Editor CTE

The classification-tree editor CTE (Figure 2) is based on the classification-tree method and supports systematic test case determination for functional testing. The two main phases of the classification-tree method - design of a classification tree and definition of test cases in the combination table - are both supported by the tool. For each phase a suitable working area is provided. To give the user optimal support, editing is done in a syntax directed and object based way. Several functions are performed automatically. These include drawing of connections between tree elements, updating the combination table after changes in the tree, and checking the syntactical consistency of table entries. The CTE also offers features which allow large-scale classification trees to be structured in order to support the test case design for large testing problems efficiently.

As test documentation plays an important role in systematic testing, the CTE offers suitable support for this activity. For example, the test case design can be documented easily by printing out the trees and tables. Furthermore, the tool can automatically generate text versions of the test cases, based on the test case definition in the combination table. For example, the text version of test case 2 of the example is shown as:

```
- coverage: yes
   - degree of coverage: one endpoint
- position of endpoint P1: line endpoint P1 located inside the rectangle
   - position of P1 in the rectangle: on one side (no corners)
      - which side: left
- position of endpoint P2: line endpoint P2 located outside the rectangle
   - position of P2 with respect to rectangle: left and above
   - distance P2 <-> rectangle: far from rectangle
- Course of line: slanting
   - direction of line: bottom right -> top left
   - gradient of line: medium.
```

On the one hand these text versions serve as documentation, on the other hand they provide a basis for the subsequent activities of software testing like the generation of concrete test data. As the CTE only supports test case determination, additional tools are required for an efficient test process which automates most of the test activities.

2. Survey of Existing Ada Testing Tools

Many different Ada testing tools are available for the examination of Ada programs (cf. [Graham et al., 1995]). This chapter summarizes the results of a detailed study of existing Ada testing tools regarding their support for the various test activities. Commercially available tools as well as non-commercial tools were included in the study in order to cover a broad spectrum of different approaches to the test. The following tools were examined on the basis of literature, vendor information, evaluation licences and demo-versions: AdaCAST (Vector Engineering), AdaTEST (Information Processing Ltd), DARTT (Daimler-Benz Aerospace AG), DEVISOR (Dassault Electronique), LDRA Testbed (Liverpool Data Research Associates Ltd), LOGISCOPE (Verilog), STW Coverage (Software Research), TAOS (University of California), TBGEN and TCMON (Testwell Oy), as well as UATL (ITT Avionics).

2.1 AdaCAST

AdaCAST [Vector Engineering, 1995] can be used for unit tests as well as for integration tests. AdaCAST integrates four components: the Test Environment Constructor, the Test Case Editor, the Test Execution Manager, and the Test Report Generator. With the help of the Test Environment Constructor the environment for the test is defined. It is determined, for example, which Ada library will be used and what the name of the test object will be. When all relevant information is available, the Environment Constructor automatically parses the unit under test, generates test driver and stubs, and compiles a complete Ada test harness. The Test Case Editor serves for giving test data. The test cases consist of data values for each formal parameter for each subprogram within the unit under test, and any dependent units that were stubbed by the Environment Constructor. With the Execution Manager the test is carried out, using any previously created test case. The Report Generator is used to construct text file reports which summarize the results of a specific test case. The main emphasis of AdaCAST is on the execution and documentation of function oriented tests. The instrumentation of the program code and a monitoring of the test coverage are not supported. A possibility of monitoring the test is to execute the test cases under control of the Compilation System Debugger. Moreover, it is not possible to specify the expected behaviour. Consequently, the test evaluation is not supported by the comparison of expected and actual values either. There is, however, the possibility of an automatic test evaluation for regression tests. Positive about this test system is that all its functions are provided via a graphical user interface.

This information was compiled on the basis of the demo-version 1.4 of AdaCAST. Test organization, stub as well as test driver generation could not be evaluated with this version.

2.2 AdaTEST

AdaTEST [IPL, 1995] is a product which has been developed to assist primarily with the unit and integration testing stages of an Ada software project. It comprises facilities in *three main areas, namely the production of test drivers and stubs, test coverage analysis and also static analysis. Consequently, the main emphasis of AdaTEST is on the activities test execution, test evaluation, monitoring, and test documentation. AdaTEST offers a script-driven approach to encourage repeatability, containing automated checks on data object values and exception behaviour. Scripts can be created in two different ways. The first one requires the tester to write his scripts as an Ada main procedure, calling the unit under test directly and using the AdaTEST Harness directives accessed via the normal Ada package mechanisms. The second approach involves the generation of the Ada main test procedure from test case data supplied by the user in the form of a Test Case Definition (TCD) file. AdaTEST scripts can be run in any validated Ada environment which confers the portability characteristic.*

AdaTEST supports about ten different forms of coverage, including code and data coverage types like statement coverage, branch coverage, and condition coverage. The software under test is instrumented automatically and test results are generated. There is a graphical option for the display of coverage against code.

2.3 DARTT

DARTT ([Gerlich and Fercher, 1993], [Gerlich, 1995]) is a tool for statistical testing. It is used primarily for the execution of stress tests for units. The test data are generated automatically for this according to the value domains of the input parameters. The test execution is also automated. As a basis for the manual test evaluation, DARTT provides an analysis of the outputs generated by the test object. In the course of this, discontinuities occurring in the output behaviour of the test object are detected and documented.

Using DARTT has its limitations. All definitions are supported except discriminant and variant records. For type “address” currently the address value “0000:0000” is returned and neither random nor incremental initialisation is done. Due to these restrictions and the general weaknesses of random testing the tool is not meant to be an alternative to other test systems but shall rather be used additionally.

2.4 DEVISOR

DEVISOR [Dassault Electronique, 1994] is a product which links functions for the debugging and the software test. The test support concentrates on test execution and test evaluation. An interpreter as well as a compiler for the DEVISOR specific test language are the most important parts. The test execution is based on a test script which can either be interpreted directly or compiled to an executable program. DEVISOR offers the possibility of drawing up the script manually but also makes it possible to generate the script automatically by using the DEVISOR-A extension. A symbol data basis generated automatically on the basis of the test object makes the access to the test object variables easier and supports test execution and test evaluation. With DEVISOR only those programs can be tested which are complete and in an executable form. DEVISOR does not offer a suitable possibility of documenting the test results.

The common criteria for structural testing as, for example, branch testing are offered optionally with the extension DEVISOR-B. Additional functions for the testing of real-time applications are provided by DEVISOR-C. The details on DEVISOR result from an evaluation of its basic version 9.0.0 and also from information provided by Dassault Electronique.

2.5 LDRA Testbed

LDRA Testbed [LDRA, 1993] is orientated towards structural testing and static analysis. The most important coverage measurements like statement coverage, branch coverage, and condition coverage are considered. Apart from the achievements in the field of monitoring, the main emphasis is on test evaluation and test documention. The extensive func-

tions for summarizing the generated results in a textual and graphical form need to be stressed here. LDRA Testbed provides all functionalities via a graphical user interface.

The basic version of LDRA Testbed restricts the possibilities of its use considerably because only one source file can be handled at a time. This does not apply when using the additional component TBset. The achievements of TBset are not contained in the graphical user interface. A more extensive test automation is offered by another component called TBrun which makes an automatic generation of the test program possible.

As a basis of this assessment, the test system LDRA Testbed, version 4.9, was evaluated including the extension TBset.

2.6 LOGISCOPE

LOGISCOPE [Verilog, 1993] combines support in the fields of static analysis and structure oriented tests. It is designed for the use subsequent to functional testing. The test functionality therefore concentrates on monitoring and test documentation. LOGISCOPE can be used at the level of the unit test as well as the integration test. As an aid for the integration test, LOGISCOPE offers a function-call graph which makes the selection of test objects easier and also shows clearly the progress of the test.

As criteria for white-box testing, statement coverage, branch coverage, and also condition coverage are available. In order to support the attainment of a complete test coverage, LOGISCOPE generates a list of program conditions for branches that have not been executed during the test so far. This list can be consulted for deriving test data which cause the execution of certain parts of the program that have not been reached before. LOGISCOPE provides a modern graphical user interface. It was evaluated in version 2.0.

2.7 STW Coverage

STW Coverage is a test coverage analysis tool from the Software Test Works (STW) tool suite which provides support for structure oriented tests. STW Coverage can be used for unit and integration tests. The functionality offered lies mainly in the fields of monitoring and test documentation. Apart from branch coverage and path coverage, the coverage of function calls is offered as a test criterion for the integration test. For documenting the coverage results, STW Coverage generates an overall view in tabular form. Additionally, the executed parts of the program can be visualized in call graphs and control-flow graphs. STW Coverage can be used via a graphical user interface but also through extensions of compiler and linker commands.

2.8 TAOS

The test system TAOS [Richardson, 1994] combines statistical tests with structural tests. Program dependencies concerning the control flow as well as the data flow of the test object are in the centre of the test with TAOS. These are first of all analysed and shall then be covered as completely as possible in the course of the test. Therefore, the functionality of TAOS supports test data generation, test evaluation on the basis of test oracles, and test documentation. The user interface of TAOS consists of several graphical editors.

The test data are either determined manually, for example on the basis of the program structure, or generated automatically by means of a test data generator. Statistical tests with simultaneous coverage analysis can therefore be carried out in addition to regular structure tests. The evaluation takes place by comparing the actual values with the values given by the test oracle. TAOS contains a set of test oracles which can be adapted to the respective tests. The test results are documented in different reports. All resulting data are stored in a repository.

2.9 TBGEN and TCMON

TBGEN and TCMON [Testwell Oy, 1996] are two separate tools which in this combination support function and structure oriented tests. Their achievements include test execution, test evaluation, monitoring, and test documentation.

The main emphasis of TBGEN is on functional testing for the unit and integration test. TBGEN consists basically of a Testbed Generator which automates the generation of a test environment. In this environment a number of TBGEN specific test instructions are provided. These instructions can be dealt with separately or can be summarized in a test script. For documenting the test results a clear report can be generated according to the needs of the user.

With TCMON structural tests such as statement coverage and condition coverage are supported. An overall view of the test coverage achieved is part of the documentation.

The given details are based on an evaluation of the demo-versions of TBGEN 5.0 and TCMON 2.3. The Testbed Generator and the report generator were not part of the demos.

2.10 UATL

The UATL (Universal Ada Test Language) is a test language developed especially for the integration test of Ada programs [Ziegeler et al., 1989]. It is orientated primarily towards the automation of function oriented, real-time closed-loop tests. With the functionality of the UATL, tests in the host environment as well as on the target system are supported. The main emphasis is on test execution, monitoring, test evaluation, and test documentation.

The UATL consists of a set of Ada packages that provide the user with a complement of reusable test functions. It also includes an interactive test manager and interactive program generation tools which simplify the test program development process. Further components are a tool for the instrumentation of the source code according to white-box test criteria, a tool for real-time data recording, and a component for data reduction analysis. A graphical presentation of the test results in different kinds of charts is also supported.

2.11 Comparison of the Testing Tools

The study shows that a lot of tools are available for the automation of test execution and monitoring. More powerful tools also offer support for test evaluation and test documentation. A systematical test case design in particular is not supported by any of the tools evaluated. None of the tools supports expected values prediction extensively. Only one tool has a component for test data selection but several tools offer functionalities for the automatic generation of random test data. Several testing tools add static analysis features. It becomes clear that the tools, as a rule, are specialized in separate areas. Table 1 summarizes the individual functionalities of the different test systems and shows for the separate tools the scope of support for every test activity.

One of the most powerful commercial tools is AdaTEST from IPL. AdaTEST's strengths are the comprehensive support for test driver generation, the high degree of automation for test execution, and the large number of different coverage criteria provided by the monitoring component. Furthermore, AdaTEST offers an interface for the integration with other tools like the CTE, the TCD file. On the basis of an existing TCD file and the source code of the unit under test, most test activities can be performed automatically.

3. Integration of CTE and AdaTEST

The aim of integrating the classification-tree editor CTE with AdaTEST is to achieve systematic and comprehensive support for the functional test of Ada units with a high degree of automation. To integrate the classification-tree editor CTE with AdaTEST, extensions to the CTE were implemented. An export interface was added to the classification-tree editor which is based on a template file containing global as well as test case specific statements. The export interface facilitates the automatic generation of TCD files corresponding to the test cases specified graphically with the classification-tree method. For each test case the specific commands from the template file are duplicated. Along *with the test case numbers, the textual representations for the test cases are generated and saved into the TCD file. Names and comments for test cases which the tester entered in the CTE can be exported to the TCD file optionally.*

	AdaCAST	*AdaTEST*	*DARTT*	*DEVISOR*	*LDRA Testbed*	*LOGISCOPE*	*STW Coverage*	*TAOS*	*TBGEN, TCMON*	*UATL*
Test Organization	++	-	+	-	+	++	+	+	-	++
Test Case Determination	-	-	-	-	-	-	-	-	-	-
Test Data Selection	+	-	++	-	-	-	-	++	-	+
Expected Values Prediction	+	+	-	+	-	-	-	-	+	+
Test Execution	++	++	++	+	+	-	-	-	+	++
Monitoring	-	++	-	+	++	++	++	++	++	+
Test Evaluation	+	++	-	++	+	-	-	++	++	+
Test Documentation	++	++	++	-	++	+	++	++	++	++

++ activity extensively supported   + activity partly supported   – activity not supported

Table 1: Functionalities of the different test systems

The generated TCD file can further be used by AdaTEST directly for the generation of the test driver. In most cases, however, the user must add test data and expected values for the individual test cases to the TCD file. This is necessary because the test cases generated with the CTE usually represent an abstract description of the input situations to be tested and do not yet contain any concrete test data or expected values. In addition to the generation of the test driver, AdaTEST also executes automatically the instrumentation of the test object. The instrumented test object and the test driver are then compiled and linked together to an executable test program which automates test execution, test evaluation, and monitoring. As a result the test documentation containing the test results is generated. In the following the use of both tools is illustrated using the example is_line_covered_by_rectangle.

3.1 Example

In the course of the test of the sample function is_line_covered_by_rectangle the systematic test case design carried out by means of the CTE is the first step. The determination of the test cases for is_line _covered_by_rectangle has already been described in chapter 1. First of all, a classification tree covering the test relevant aspects is developed. Afterwards the test cases are determined by combining classes of different classifications in the combination table. 49 test cases were defined for the test of the sample function. Figure 2 shows a part of this classification tree and the combination table belonging to it.

For the next step a TCD file is generated by the CTE. For doing so the user enters the name of the TCD template to be used for AdaTEST into a dialogue box of the CTE. He then selects the Export operation of the CTE’s file menu. The TCD file produced contains instructions in the AdaTEST specific Test Case Definition Language as specified in the corresponding template file. For each test case the TCD file contains the specification and a framework for calling the test object and giving the test data and the expected values. The order of the test cases is analogous to that in the CTE. The following paragraph shows *a part of the generated TCD file along with the instructions generated automatically for the first test case.*

```
%%TEST_CASE 1;

-- Test Case description
 %%PURPOSE "- coverage: yes";
 %%PURPOSE "  - degree of coverage: complete";
 %%PURPOSE "- position of endpoint P1: line endpoint P1 located inside the rectangle";
 %%PURPOSE "  - position of P1 in the rectangle: inside the rectangle";
 %%PURPOSE "- position of endpoint P2: line endpoint P2 located inside the rectangle";
 %%PURPOSE "  - position of P2 in the rectangle: inside the rectangle";
 %%PURPOSE "- Course of line: slanting";
 %%PURPOSE "  - direction of line: bottom left -> top right";
 %%PURPOSE "  - gradient of line: medium";

%%CALL  graphics.is_line_covered_by_rectangle, " ";

%%INPUTS
     rectangle :=  ;
     line := ;

%%OUTPUTS
      ATS_RETURN := ;
```

As a next step the test data and the expected values for each test case are added by editing the TCD file. In case the test data and the expected values have been entered completely, the test driver can be generated automatically from the TCD file by means of the AdaTEST script generator. Test execution, coverage analysis, test evaluation and documentation will then take place automatically in the framework of the AdaTEST functionality.

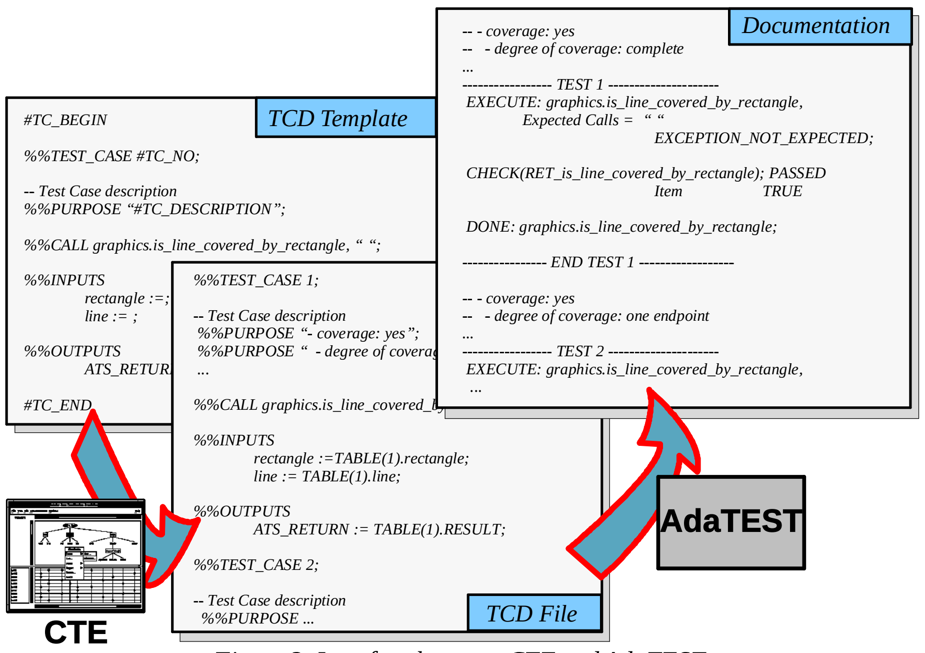
By executing the 49 test cases no error was detected in is_line_covered_by_rectangle. 100 % branch coverage as well as 100 % operand coverage were achieved with this test which hints at the good quality of the functional test. A part of the documentation generated with AdaTEST is shown in Figure 3. Figure 3 also shows parts of the three main files which were constructed or generated automatically combining CTE and AdaTEST. The interrelation between the specified test cases, the test run, and the test results is documented for the first test case.

4. Practical Experience

The classification-tree editor CTE has been used successfully in various projects for the systematic test case design for more than four years now (cf. [Grochtmann and Wegener, 1995]). In comparison with previous tests, perceptible savings could be achieved: the number of test cases could be reduced and their quality has been improved considerably. Thus the cost for the entire tests has been diminished substantially. Because of the positive user acceptance, a product version of the CTE has been developed and can be obtained from ATS (Automated Testing Solutions) since 1995.

The CTE-AdaTEST integration has already been successfully applied to several real world examples including aerospace applications and internal software products of Daimler-Benz Research. An application in the field of aviation was a rate limiter. For this example 28 test cases were determined systematically with the CTE. In the course of the test carried out on the basis of these test cases, 100 % coverage was measured for all structural test criteria supported by AdaTEST. In a further aerospace example a function of the complex COLUMBUS SW Development Environment (SDE) was examined. 16 test cases were determined. The execution of these test cases led to 97 % statement coverage. At the same time it was discovered that the exception coverage amounted to only 33 %. Those parts of the program that had not been reached could, however, be easily identified with the help of the clear presentation of the coverage analysis results in the test documentation. On the basis of the program information additional test cases could be determined which then led to 100 % exception coverage. One part of the functionality of this test object is the processing and the comparison of dates. In the course of the test it turned out that the unit under test did not recognize that some dates like April 31 were false.

The combined use of CTE and AdaTEST proved to be very helpful in the sample applications mentioned above. The quality of the tests and the number of test cases are influenced



```
#TC_BEGIN

%%TEST_CASE #TC_NO;

-- Test Case description
%%PURPOSE "#TC_DESCRIPTION";

%%CALL graphics.is_line_covered_by_rectangle, " ";

%%INPUTS
        rectangle :=;
        line := ;

%%OUTPUTS
        ATS_RETUR

#TC_END
```

```
%%TEST_CASE 1;

-- Test Case description
 %%PURPOSE "- coverage: yes";
 %%PURPOSE "  - degree of coverag
 ...

%%CALL graphics.is_line_covered_b

%%INPUTS
        rectangle :=TABLE(1).rectangle;
        line := TABLE(1).line;

%%OUTPUTS
        ATS_RETURN := TABLE(1).RESULT;

%%TEST_CASE 2;

-- Test Case description
  %%PURPOSE ...
```

```
-- - coverage: yes
--   - degree of coverage: complete
...
----------------- TEST 1 ---------------------
 EXECUTE: graphics.is_line_covered_by_rectangle,
          Expected Calls =   " "
                                  EXCEPTION_NOT_EXPECTED;

 CHECK(RET_is_line_covered_by_rectangle); PASSED
                                  Item               TRUE

 DONE: graphics.is_line_covered_by_rectangle;

---------------- END TEST 1 ------------------

-- - coverage: yes
--   - degree of coverage: one endpoint
...
----------------- TEST 2 ---------------------
 EXECUTE: graphics.is_line_covered_by_rectangle,
  ...
```

Figure 3: Interface between CTE and AdaTEST

considerably by the systematic test case determination. Moreover, the functionalities of AdaTEST offer an appropriate environment for an efficient test execution. The coverage analysis provided by AdaTEST offers a further evaluation criterion for the test. The systematically generated test cases, the test results, and the results of the coverage analysis summarized in the test documentation give a suitable overall view of the test which also allows to detect errors quickly. Furthermore, confidence in the function of the test object is established provided that no errors were found.

5. Conclusion

A number of professional tools exists for the unit testing of Ada programs. None of these tools, however, offers overall support for all test activities. Moreover, the tools evaluated lack methodological support for the important activity of functional test case design. Most of the tools concentrate on structure oriented test case determination on the basis of the results obtained from coverage analysis. In order to fill this gap the classification-tree editor CTE can be combined with existing tools.

The CTE is a graphical editor for the functional test case determination and supports the application of the classification-tree method. It has already proved very worthwhile in many different areas of application in the industrial practice and also has a flexible interface which facilitates the combination with other tools. An integration of the CTE with AdaTEST has been presented in this paper. An integration of the CTE with the test system TESSY for the testing of C programs is described in [Wegener and Pitschinetz, 1995].

The combination of the CTE with IPL's AdaTEST covers most of the activities which are essential to the unit testing and integration testing of Ada programs. Due to the methodological basis and the high degree of automation, the combination of both tools leads to systematic, well-documented, and efficient test procedures. Future work will focus on extensions for test data selection and expected values prediction as well as the automatic generation of test cases and test data. Furthermore, an integration of the CTE with the Test

Manager from SQA is in preparation to support systematic test case design also for the examination of applications with graphical user interfaces.

Acknowledgements

We are grateful to the following persons for their assistance with the different Ada test systems: Dr. Vieser and Mr. Hennell (LDRA Testbed), Mr. Gerlich (DARTT), Mrs. Buisson and Mr. Lernoud (DEVISOR), Mr. McCaffrey (AdaCAST), and finally Mr. Poutanen (TBGEN and TCMON).

References

Dassault Electronique (1994). DEVISOR System Tutorial and User Manual, DEVISOR System Test Language Reference Manual Version 9.4, DEVISOR Software debug and test system, 1994, Dassault Electronique, Saint-Cloud, France.

Gerlich, R. (1995). DARTT User's Manual Version 2.0, 1995, Dornier GmbH, Friedrichshafen, Germany.

Gerlich, R., and Fercher, G. (1993). A Random Testing Environment for Ada Programs. Contribution to Forth EUROSPACE symposium on "Ada in Aerospace", 8 - 11 November 1993, Brussels, Belgium.

Graham, D., Herzlich, P., and Morelli, C. (1995). CAST Report - Computer Aided Software Testing. 1995, Cambridge Market Intelligence, London, UK.

Grochtmann, M., and Grimm, K. (1993). Classification Trees for Partition Testing. Software Testing, Verification and Reliability, Vol. 3, No. 2, pp. 63 - 82.

Grochtmann, M., and Wegener, J. (1995). Test Case Design Using Classification Trees and the Classification-Tree Editor CTE. Proceedings of Quality Week '95, 30 May - 2 June 1995, San Francisco, USA.

IPL (1995). AdaTEST Harness Version 3.0 User Guide & Reference Manual, AdaTEST Analysis Version 3.0 User Guide & Reference Manual, 1995, Information Processing Ltd., Bath, UK.

LDRA (1993). LDRA Testbed Technical Description, 1993, Liverpool Data Research Associates Ltd., Liverpool, UK.

Ostrand, T., and Balcer, M. (1988). The Category-Partition Method for Specifying and Generating Functional Tests. Communications of the ACM, 31 (6), 1988, pp. 676 - 686.

Richardson, D.J. (1994). TAOS: Testing with Analysis and Oracle Support. ACM SIGSOFT Software Engineering Notes: Proceedings of the 1994 International Symposium on Software Testing and Analysis. 1994, Seattle, Washington, USA.

Testwell Oy (1996). What's new in TBGEN 5.0 ?, 1996, Testwell Oy, Tampere, Finland.

Vector Engineering (1995). AdaCAST Product Overview, 1995, Vector Engineering, North Kingstown, USA.

Verilog (1993). LOGISCOPE Ada Analyzer 2.0 Reference Manual, 1993, Verilog SA, Toulouse, France.

Wegener, J., and Pitschinetz, R. (1995). TESSY - An Overall Unit Testing Tool. Proceedings of Quality Week '95, 30 May - 2 June 1995, San Francisco, USA.

Ziegler, J., Grasso, J.M., and Burgermeister, L. (1989). An Ada Based Real-Time Closed-Loop Integration and Regression Test Tool. 1989, ITT Avionics, Washington, USA.