

4

AD-A218 808

REPORT SSD-TR-89-81

DTIC FILE COPY

An Analysis of
"The Definition of a Production Quality Ada Compiler"
Volume I

Prepared by
B. A. PETRICK
Software Development Department
S. J. YANKE
Systems Software Engineering Department
Engineering Group
The Aerospace Corporation
El Segundo, CA 90245

13 March 1989

Prepared for
SPACE SYSTEMS DIVISION
AIR FORCE SYSTEMS COMMAND
Los Angeles Air Force Base
P.O. Box 92960
Los Angeles, CA 90009-2960

DTIC
ELECTE
MAR 08 1990
S E D

APPROVED FOR PUBLIC RELEASE;
DISTRIBUTION UNLIMITED

90 03 08 020

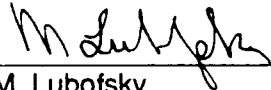
This report was submitted by The Aerospace Corporation, El Segundo, CA 90245, under Contract No. F04701-88-C-0089 with Space Systems Division, P.O. Box 92960, Los Angeles, CA 90009-2960. It was reviewed and approved for The Aerospace Corporation by R. D. Hefner, Director, Software Development Department, Engineering Group. Capt. John Brill, SSD/ALR, was the project officer for the program.

This report has been reviewed by the Public Affairs Office (PAS) and is releasable to the National Technical Information Service (NTIS). At NTIS, it will be available to the general public.

This technical report has been reviewed and is approved for publication. Publication of this report does not constitute Air Force approval of the report's findings or conclusions. It is published only for the exchange and stimulation of ideas.



A. E. Stevens, Lt. Col., USAF
SSD/ALR



M. Lubofsky
Senior Engineering Specialist
Computer Resources Management
and Standards Office

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a REPORT SECURITY CLASSIFICATION Unclassified			1b RESTRICTIVE MARKINGS			
2a SECURITY CLASSIFICATION AUTHORITY			3 DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited.			
2b DECLASSIFICATION/DOWNGRADING SCHEDULE						
4 PERFORMING ORGANIZATION REPORT NUMBER(S) TR-0089(4902-03)-1 Vol I			5 MONITORING ORGANIZATION REPORT NUMBER(S) SSD-TR-89-81			
6a NAME OF PERFORMING ORGANIZATION The Aerospace Corporation Engineering Group		6b OFFICE SYMBOL <i>(If applicable)</i>	7a NAME OF MONITORING ORGANIZATION Air Force Systems Command Space Systems Division			
6c ADDRESS (City, State, and ZIP Code) 2350 E. El Segundo Blvd. El Segundo, CA 90245			7b ADDRESS (City, State, and ZIP Code) Los Angeles Air Force Base Los Angeles, CA 90009-2960			
8a NAME OF FUNDING/SPONSORING ORGANIZATION		8b OFFICE SYMBOL <i>(If applicable)</i>	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F04701-88-C-0089			
8c ADDRESS (City, State, and ZIP Code)			10 SOURCE OF FUNDING NUMBERS			
			PROGRAM ELEMENT NO	PROJECT NO	TASK NO	WORK UNIT ACCESSION NO
11 TITLE (Include Security Classification) An Analysis of "The Definition of a Production Quality Ada Compiler" Volume I						
12 PERSONAL AUTHOR(S) B. A. Petrick, S. J. Yanke						
13a TYPE OF REPORT		13b TIME COVERED FROM TO		14 DATE OF REPORT (Year, Month, Day) 1989 March 13		15 PAGE COUNT 58
16 SUPPLEMENTARY NOTATION						
17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)			
FIELD	GROUP	SUB-GROUP	Ada compiler, selection; Ada compiler, procuring; Ada compiler, specifications; (cont.)			
19 ABSTRACT (Continue on reverse if necessary and identify by block number) This report outlines a procedure for using "The Definition of a Production Quality Ada Compiler", SD-TR-87-29, as the basis for determining if an Ada compiler is of production quality. The report describes the development of a test suite from the requirements set forth in SD-TR-87-29, as well as the results of applying this test suite to two validated Ada compilers. An analysis of SD-TR-87-29 from creating and applying the test suite has also been provided.						
20 DISTRIBUTION/AVAILABILITY OF ABSTRACT <input type="checkbox"/> UNCLASSIFIED/UNLIMITED <input checked="" type="checkbox"/> SAME AS REPORT <input type="checkbox"/> DTIC USERS			21 ABSTRACT SECURITY CLASSIFICATION Unclassified			
22a NAME OF RESPONSIBLE INDIVIDUAL Capt. John Brill			22b TELEPHONE (Include Area Code) (213) 643-2532		22c OFFICE SYMBOL SSD/ALR	

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE

18. SUBJECT TERMS (Continued)

Ada compiler, evaluating; Ada Compiler, requirements;
Ada compiler, test suite; production quality Ada compiler;
Project Ada compiler.

Ada programming language,
Computer Program Documentation (g.c.)

SECURITY CLASSIFICATION OF THIS PAGE
Unclassified

Acknowledgment

This study was sponsored by Space Systems Division Directorate for Computer Resources (SSD/ALR). Funding for the effort was provided by the Air Force Computer Resource Management Technology Program, Program Element (PE) 64740F, Project 2526, Software Engineering Tools and Methods.

Program Element 64740F is the Air Force engineering development program to develop and transfer into active use the technology, tools, and techniques needed to cope with the explosive growth in Air Force systems that use computer resources. The goals of the program are to: (a) provide for the transition of computer system developments from laboratories, industry, and academia to Air Force systems; (b) develop and apply software acquisition management techniques to reduce life-cycle costs; (c) provide improved software design tools; (d) address the various problems associated with computer security; (e) develop advanced software engineering tools, techniques, and systems; (f) support the implementation of high-order languages, e.g. Ada; (g) address human engineering for computer systems; and (h) develop and apply computer simulation techniques for the acquisition process.

Accession For	
NTIS GRA&I	<input checked="" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	



Executive Summary

The Aerospace Corporation released *The Definition of a Production Quality Ada Compiler* (SD-TR-87-29), herein referred to as the *Definition*, in March of 1987. The intended purpose of the *Definition* was to provide a quantifiable definition of the term "production quality".

The *Definition* specified minimal and desirable requirements of an Ada compiler that are beyond those required for validation. These include performance, capacity limits, user-interface, reliability, documentation, and certain language features left to the discretion of the compiler builder. It was intended that these requirements be used in validating a vendor's claim of production quality for a compiler.

At the outset of a software project, it is extremely important to know that the proposed language compiler is of production quality. A compiler is not only a source of single point failure in the extreme; it is also a potential major bottleneck to achieving progress during the project's coding and testing phase.

Because it was recognized that quantifying such a definition is very difficult, the following study, funded by the Air Force Computer Resources Management Technology Program Element, PE 64740F, has been conducted for Space Systems Division Directorate for Computer Resources, SSD/ALR, to provide an analysis of the *Definition*. It describes how the requirements in the *Definition* can be applied to an Ada compiler. In particular, this report outlines a procedure for using the *Definition* as the basis for determining if an Ada compiler is of production quality. Problems and suggestions uncovered while creating and applying the procedure to two Ada compilers are also presented.

The *Definition* contains 144 separately discernible requirements in 45 sections. A test for each of the identified requirements has been developed in the *Production Quality Ada Compiler* (PQAC) test suite. Two compilers were assessed as part of the mechanism used to evaluate the *Definition*. The PQAC test suite was applied to the DEC VAX V1.4 and Telesoft TeleGen2 V3.15 Ada compilers which ran on a VAX 8600 under VAX VMS 4.7.

Fourteen of the requirements in the *Definition* are designated as being fundamentally essential for a production quality Ada compiler. Two of these requirements address compilation speed issues. Seven of these requirements are concerned with external manuals or documents. The remaining five requirements address object code size, object code speed, compiler error rate, field testing, and compiler validation. At the time our analysis was performed, the DEC VAX failed one of the essential requirements for compilation speed. The Telesoft failed six essential requirements: object code size, object code speed, compiler error rate, field testing, and both compilation speed requirements. It must be emphasized that changes and improvements can be expected from both compilers,

and that the essential requirements should be reassessed by anyone considering either compiler.

In general, different applications may identify certain requirements as being more important than others. In an effort to handle these conflicts, the ability to independently weight each of the requirements was created. As an example, if it is determined that a compiler need only satisfy the essential requirements, a weight of 1 could be given to these requirements and a weight of 0 to the rest.

For the purposes of this report, however, the requirements were assigned weights from 1 to 10. It is felt that this range was adequate for reflecting the difference in magnitude between a single language feature requirement and an essential requirement. In addition to the essential requirements, those requirements concerned with run-time performance, program development, or adherence to a standard were given a higher weight.

The results obtained showed a marked difference in the performance of these two compilers. Using the weighting scheme devised for the test suite, the DEC VAX and Telesoft compilers achieved a success rate of 88% and 67%, respectively. These results are dependent on the weights chosen and the validity of the requirements themselves. If it is determined that the essential requirements should contribute more heavily to a compiler's rating, the weights for those requirements could be increased.

Because of the small sample size of only two compilers, it is difficult to interpret the level of production quality these statistics represent. It may be that this single performance statistic is most useful when used to compare Ada compilers against each other, and should not be used as a flat measure of a compiler. The value obtained should be viewed as an indicator of a compiler's production quality. This rating may be used to direct attention to failed tests that need to be examined further.

As the results illustrate, the *Definition* provides a useful vehicle for measuring the extent to which the tested compilers are of production quality. Since the PQAC test suite was newly developed during the course of this project, a number of problems with implementing such a test suite from the *Definition* were identified. These problems with suggested solutions are discussed in this report. In addition, periodic review by the Ada community is recommended as a mechanism for keeping the *Definition* current as compiler technology advances.

Table of Contents

Executive Summary	1
1. Introduction	5
2. PQAC Test Suite Development	7
2.1 Requirement Analysis	7
2.2 Test Representation	7
2.3 Test Method	8
2.4 Support Software	9
2.5 Special Features	10
2.6 Test Execution and Evaluation	11
3. Result Interpretation Using Weights	12
3.1 Essential Requirements	12
3.2 Weighting Scheme	13
3.3 Weights Chosen	13
3.4 Result Interpretation	15
4. DEC VAX and Telesoft Compiler Evaluations	16
4.1 PQAC Statistics	16
4.2 DEC VAX Results	17
4.3 Telesoft Results	18
5. The <i>Definition</i> Analysis	19
5.1 Requirement Review	19
5.2 Deficiencies	19
5.3 Summary	20
Acronyms	22
Appendices	23
A. Test and Requirement Cross Reference	24
B. PQAC Test Weights	38
C. DEC VAX Ada Result Summary	42
D. Telesoft VAX Ada Result Summary	46
E. Additional Comments	50
F. Test Examples	54
F.1 Code Expander Description	54
F.2 Test Description	56
G. PQAC Test Suite - Volume II	58

Tables

1a.	Appendix B Weighting Scheme (500 Possible)	16
1b.	No Weighting Scheme - Equally Weighted (144 Possible)	16
2.	Failure Points Using Appendix B Weighting Scheme	17

1. Introduction

The *Definition of a Production Quality Ada Compiler*, herein referred to as the *Definition*, specifies minimal and desirable requirements of a production quality Ada compiler. The following study describes how the requirements in the *Definition* can be applied to an Ada compiler.

The procedure outlined in this report has been applied to the DEC VAX V1.4 and Telesoft TeleGen2 V3.15 Ada compilers running on a VAX 8600 under VAX VMS 4.7. Problems and suggestions uncovered while creating this report have been recorded.

In both the *Definition* and this report, the meaning of the term *compiler* has been expanded to include the Ada Programming Support Environment (APSE). It is recommended that anyone reading the following report have a copy of the *Definition* available for reference.

The development of the *Production Quality Ada Compiler* (PQAC) test suite is described in Chapter 2. A separate test for each of the requirements in the *Definition* has been developed. Appendix A contains a cross reference between the PQAC tests and the requirements. This Appendix may be consulted to resolve any reference to a test number in this report.

The PQAC test suite is designed to be independent of the compiler and host/target architecture under test. Tables in the test support software capture dependent features of the interfaces such as file name, system time, and compiler option syntax. This allows the test suite to be easily rehosted for different compilers and host/target architectures. In addition, portions of a test may be designated as applicable to a specific compiler. As each configuration of compiler and host/target architecture is included in the support software, it becomes a permanent part of the test suite domain.

Since not all of the tests are of equal importance, a capability for independently weighting each of the individual requirements in the *Definition* has been developed. Using this capability, those requirements designated by the *Definition* as essential may be assigned a higher weight. Requirements determined to be important to specific applications may also be assigned a higher weight. Appendix B contains a table of the weighting scheme used in this report. The development of the weighting scheme used is described in Chapter 3.

The PQAC test suite was applied to the DEC VAX and Telesoft Ada compilers. A summary of the results is given in Chapter 4 with a list of the tests passed for each compiler given in Appendices C and D, respectively. A more detailed analysis of the failed requirements may be obtained by reviewing the test data and results. It should be noted that the evaluation of these two compilers is secondary to the analysis of the *Definition* itself. The validity of the

compiler evaluations are dependent on the analysis of the *Definition* and on the weighting scheme chosen. A different weighting scheme could alter the outcome of the compiler performance analysis.

An effort has been made to identify any untestable, inconsistent, and missing requirements in the *Definition*. Appendix E contains comments and suggestions not stated elsewhere in the report arising from the development and application of the test suite.

A considerable body of test support software (1500 Ada source lines) was created to facilitate the execution of the test suite. Appendix F contains test examples and illustrations of several test support software features. A fully detailed User's Guide will be completed as the test suite matures.

2. PQAC Test Suite Development

The *Production Quality Ada Compiler (PQAC)* test suite has been developed to provide a method of determining if a compiler adheres to the requirements in the *Definition*. The PQAC test suite consists of the tests corresponding to each of the requirements in the *Definition* as well as the support software used to execute the tests and gather results.

2.1 Requirement Analysis

The *Definition* is composed of a non-homogeneous set of requirements. Some are concerned with Ada language features and capacities while others are APSE requirements for external manuals and documents. Some requirements are very precise while others are prey to interpretation. The method by which each of the requirements can be tested is affected by this non-homogeneity.

The requirements can be split into three main categories depending on their identified test method:

1. (65% of PQAC) Requirements that can be tested using unmodified Ada code regardless of the compiler or host/target architecture under test (e.g., verifying that case statements can be nested to a depth of 64 levels).
2. (5% of PQAC) Requirements that can be tested using Ada code requiring changes depending on the compiler or host/target architecture being tested. These requirements deal with constructs whose implementation is left to the compiler vendor or dependent on the host environment (e.g., verifying that the compiler shall provide predefined types in the Ada package STANDARD for all integer and floating-point types provided by the target computer).
3. (30% of PQAC) Requirements that cannot be tested using Ada code. These generally require some form of manual action (e.g., checking for the existence of a User's Manual).

A uniform approach encompassing these categories has been adopted for executing and gathering results from each of the tests.

2.2 Test Representation

Each separate requirement in the *Definition* has been assigned a unique test number. A complete list of these numbers is given in the Test and Requirement Cross Reference of Appendix A. A test number such as *T061208* refers to the requirement in section 6, subsection 12, requirement 8 of the *Definition*.

There exists a file for each test containing all of the information needed by the support software to correctly perform the test. This information includes:

- The test number.
- A reiteration of the requirement.
- A description of the proposed test method.
- A note of any implementation dependencies in the test.
- The test options.
 - file name and compilation name information
 - special action identifier
 - compiler options
- The Ada code needed for the test, if any.

In this manner, all three categories of test method described in the previous section may be uniformly represented. For those requirements that are testable using Ada code, the test code is included in the test file. The test method description for these requirements states how to interpret the output generated from the test. The Ada code section is omitted for those requirements with a manual test method. The test method description for these requirements simply states the manual procedure to be performed.

Any compiler or architecture implementation dependencies are highlighted in each test. This allows tests requiring changes to be quickly identified and modified for new configurations. Where appropriate, guidelines for altering the test for a new configuration are included within the test.

To allow for such dependencies, any part of a test may be designated as specific to a particular compiler. For example, certain pragmas are allowed to have an implementation defined syntax. The support software allows several versions of such a pragma to be embedded in the same test code identified with a descriptor for the appropriate compiler. In this manner, test information about a single requirement is contained in a single place. If new code specific to a compiler needs to be developed, it can simply be inserted into the test while preserving the existing code segments.

2.3 Test Method

For many requirements, only the simplest or minimal condition sufficient to verify the requirement is tested. This approach was adopted since the more complicated a test is, the greater chance it has of failing for some reason not related to the requirement being tested.

For example, in *T030101*, the requirement for allowing 2048 library units in a program library is tested. The 2048 library units used in the test each consisted of a package containing a single constant declaration. If 2048 very complicated packages had been used, the possibility of the test aborting because of a storage capacity error not related to the given requirement would become great. Likewise, in *T030501*, 64 parameters of type integer were used to test the requirement for allowing 64 formal parameters in a subprogram. The requirement was not tested using more complex parameter types such as records or arrays.

In addition, each test was constructed to be independent of the success or failure of any other. Although some of the requirements are dependent on each other, the execution of the tests themselves can proceed in any order, albeit being possibly redundant.

For those tests consisting solely of a manual test procedure, the instructions were made as explicit as possible to ensure the same interpretation by independent testers. However, most of these tests do not lend themselves to a rigid or clearly defined test method. For many of these tests, the method is simply "Check for Existence."

If a requirement is determined to be not applicable to a particular compiler or host/target architecture, then the test for this requirement may be removed from the test suite results. For example, *T080800* need only be applied when a compiler is being newly developed for a DOD contract.

2.4 Support Software

This section presents a brief discussion of the main features of the PQAC test suite support software. An effort has been made to create a compiler and host environment independent test suite in order to minimize the effort of transition to a new configuration. Appendix F contains a more detailed explanation of the support software and its functionality.

Tables have been constructed to capture compiler and host dependent information such as file name formats, system time formats, and compiler option syntax. These tables may be appended with additional entries as compilers and hosts are included in the test suite domain. As each test configuration is incorporated into the test suite, the test bed is expanded while retaining all of its previous functionality.

A standard set of compiler options has been defined. Thus, each test file does not have to be modified for compilers with a different option syntax. Options to provide a compiled or machine code listing, optimize for time, optimize for space, and perform syntax checking are available. Each compiler has an entry in a table that provides a translation from the standard option to the actual option required by that particular compiler. For example, the

defined standard option "OPTIMIZE_TIME" contains an entry for the VAX compiler "/OPTIMIZE=TIME", and for the Telesoft compiler "/OPTIMIZE=ALL".

A minimal set of operating system primitives was also defined. They consist of simple command names for all the discrete actions provided by the operating system that are needed during the execution of a test. They include LIST, PRINT, DELETE, COMPILE, LINK, EXECUTE, TIME, and SIZE. Other test options exist for invoking special actions such as compilation and execution speed timing, and measuring object code size. Each of these special actions can be achieved using only the defined operating system primitives.

To execute a test, its test representation file is first parsed and a script file is created. Any designated section of the test not targeted for the compiler under test is simply ignored. The script file produced contains only the operating system primitives described in the previous paragraph. A simple command interpreter for the appropriate operating system is then used to perform the required steps in a sequential manner.

2.5 Special Features

Several utilities have been created for use by the test suite support software. A program for counting the number of Ada source lines in a program has been developed. This has been used for computing compilation speed in lines/minute. Additionally, programs for computing elapsed time for both internal execution times and operating system level actions have been created. It is also possible to compare FORTRAN and Ada programs for both code size and code execution speed.

The most innovative feature of the support software is the capability for expanding embedded Ada code segments. Testing several of the capacity requirements involves the use of very large Ada programs. For example, in *T030101*, 2048 library units are required. In *T030103*, a program containing 2,500,000 Ada source statements must be used. The size of the test code needed for these examples would make the tests unmanageable and prohibitive. A code expander tool has been created that takes Ada code templates and expands them any number of iterations while producing correct Ada code.

Templates for expansion contain embedded special commands that the code expander recognizes. The central feature used by the code expander is a simple loop construct that may be nested. Each loop body may be replicated any number of times with each iteration producing similar but semantically unique code fragments. Before compilation, the code expander is applied to a test. The resultant code is then compiled and subsequently deleted. In this manner, large bodies of code are only present while they are being compiled, and then only one at a time.

Using the code expander on test *T030101* dropped the size of the code from 4096 source lines (conservatively) to only 25 lines of test code including the code expander syntax. The code expander tool proved to be very useful on even moderately sized tests. Of the 92 tests in the PQAC using Ada code, 48 of them also use the code expander. A detailed description, including examples, of the code expander tool is given in Appendix F.

2.6 Test Execution and Evaluation

After a test file is parsed, the script file produced will contain commands to carry out the following actions:

- Set up the Ada program library environment if needed.
- Use the Code Expander if needed.
- Compile any Ada text using the appropriate compiler options.
- Link and execute any executable code.
- Perform any special action required by the test.
- Delete all temporary and library files created by the test.
- Output the results of the test.

As each script file is processed by the command interpreter, the output is saved in a separate file. This output includes a reiteration of most of the information present in the original test file appended with status and error information. If a compiler listing or machine code listing is called for, it is also included in the output. Finally, if the test contains any code to be executed, the output of the code is also retained here. In this manner, all of the tests may be processed at one time with the results being saved for evaluation.

Included in the test method is a description of how to determine if a test passed. Some of the test evaluation methods are:

- A success or failure message generated by the test.
- The success or failure of a compilation.
- Examination of the assembly code listing for specified information.

The output from each test contains all the information needed to evaluate the success or failure of the test with the exception of any manual procedures that must be completed. Many tests automatically record their outcome in the test suite database. Users are notified of tests that require manual intervention. The third evaluation method listed above is a combination of automatic and manual testing. The assembly listing is automatically generated for those tests that require it, but the listing must be manually examined.

3. Result Interpretation Using Weights

There are 149 requirements identified in the *Definition*. For each of these requirements, there is a corresponding test in the PQAC test suite. Five of these requirements are definitions, but are included in the test suite for completeness. The previous section describes how the test suite can be applied to a compiler. A pass or fail determination for each of the individual tests may then be made. This section describes how these individual results may be interpreted.

3.1 Essential Requirements

Fourteen of the requirements in the *Definition* are designated as being fundamentally essential for a production quality Ada compiler. These 14 requirements address issues such as compilation speed, object code size, object code speed, compiler error rate, field testing, compiler validation, and external manuals or documents quality. The implication of a compiler failing to meet an essential requirement is not specified by the *Definition*. For example, in *T020401*, if a compiler achieves only 190 lines/minute/MIP compilation speed instead of the required 200, is this compiler to be excluded from being considered production quality even if it satisfies the remaining requirements?

It may be misleading to designate certain requirements as essential. For many applications, there are constructs being tested from Chapter 13 of MIL-STD-1815A that are as equally important as the essential requirements. For example, representation clauses may be indispensable for the correct execution of some embedded real time systems. In such a system, compilation speed may not be a factor at all. In general, the importance of each of the requirements for different applications is not static.

These 14 requirements also represent some of the most difficult to verify. Demonstrating that a compiler error rate is decreasing with time is virtually impossible to do in the framework of a test suite. The requirements for manuals and external documents are also difficult to test. If the compiler vendor uses a document title different from the one quoted in the requirement, it may be difficult to determine if such a document exists. Even if the document exists, verifying that it is adequate is a complicated and subjective process.

The essential requirement for a compiler validation summary report (see *T080100*) may not be needed. There already exists an essential requirement for the validation of an Ada compiler (see *T070100*). And once a compiler has been validated, the validation summary report becomes public domain. In addition, it may be desirable to apply the PQAC test suite to an unvalidated Ada compiler for comparison with other validated compilers. Currently, this would not be possible since an unvalidated compiler would fail these two tests.

It should be noted that the PQAC test suite is not aimed specifically at measuring compiler performance. There already exist test suites to perform that function. This is evidenced by the fact that the requirements are stated as a pass or fail option. For example, in *T020401*, simply knowing whether a compiler has passed the test may not be sufficient. Suppose compiler X compiles at a rate of 199, compiler Y at a rate of 200, and compiler Z at a rate of 1000 lines/minute/MIP. The fact that X and Y are comparable but Z is superior is not evident from the pass/fail test result as X would fail but Y and Z would pass. A further analysis of the test results is needed.

Although the essential requirements are important, they alone do not define production quality. It is also possible for a compiler to be of production quality for use on some projects, but not on others. The weighting scheme described in the next section has been developed for differentiating between the essential and less important requirements.

3.2 Weighting Scheme

As previously established, all requirements in the *Definition* are not of the same importance in determining production quality for an Ada compiler. In addition, the importance of some of the requirements may not be static for all applications. For these reasons, a capability for weighting each of the requirements has been created.

After the test suite has been applied to a compiler, a pass/fail percentage for each individual test is available. A partial pass for a test is possible, since some of the requirements contain multiple parts. A user modifiable table is used to hold the weights for each of the tests along with instructions for awarding points to partial passes. A user may specify whether or not he wants a partial pass to receive partial points. Points for each test can then be awarded depending on the pass percentage and total weight for the test.

The points awarded for each test are automatically accumulated, producing a test suite pass total. Tests that are deemed not applicable to a compiler or host are given a weight of 0, as are the five definitions. The test suite pass total is then divided by the total number of available points to arrive at a test suite pass percentage between 0% and 100%.

3.3 Weights Chosen

No assignment of weights is going to yield a totally accurate measure for determining production quality in all cases. For the most part, however, if the weights are carefully chosen, the result obtained will be a good indication of the compiler's level of production quality. This measure will be more accurate than one obtained without weights.

Two independent sources were tasked to weight the tests. The criteria used with examples of each type are:

- Run-time performance
 - executable code size
 - executable code speed
- Importance to program development
 - presence of user's manual
 - presence of a symbolic debugger
 - ability to provide a dependency listing
 - Invokable from batch file or interactive
 - compilation speed
- Promotes adherence to a standard
 - MIL-STD-1815A Chapter 13 support
 - predefined pragma support
 - following I/O conventions
- High frequency of occurrence
 - compilation speed
 - invokable from batch file or interactive

Since no two assignments of weights by independent sources are going to be identical, these two schemes were combined to form a single, possibly less biased, scheme. Appendix B contains a complete list of the weights chosen with additional instructions for determining how to award points for partial passes.

Each test has been assigned a weight in the range from 1 to 10. A finer granularity would probably not add any accuracy to the weighting scheme, since the assignment of weights is somewhat subjective. It is felt that this range is adequate for reflecting the difference in magnitude between a single language feature requirement and the more important essential requirements. The combined possible weight total for the scheme developed is 500 points.

3.4 Result Interpretation

The weighting scheme method described in this section will produce a test suite pass percentage for each compiler between 0% and 100%. This pass percentage should not be used as an absolute measure of an Ada compiler. A quantity such as "production quality" can never be represented exactly as a single value. Examining what tests failed for each compiler is needed to provide a complete analysis.

The result obtained is also dependent on a number of factors, including the host/target architecture and weighting scheme chosen. For example, if it is determined that the essential requirements should contribute more heavily to a compiler's rating, the weights for those requirements could be increased.

The test suite pass percentage is good, however, for providing an overall measure of a compiler. It may best be used in comparing Ada compilers against each other. It may also be used to quickly identify compilers whose performance is suspect and in need of further analysis. If a compiler has achieved a rating of 100%, then it certainly must be a very strong candidate for being considered production quality. Likewise, if a compiler is rated at a very low percentage (perhaps less than 50%), it is suspect. For those compilers whose pass percentages are within a few percent of each other, a further analysis of what tests failed would be needed to distinguish them.

4. DEC VAX and Telesoft Compiler Evaluations

Summaries of the results for the DEC VAX and Telesoft compilers are given in Appendices C and D, respectively. All of the output generated from the PQAC test suite is too voluminous to reproduce here. Appendix B contains the weighting and point assignment scheme used by the test suite.

4.1 PQAC Statistics

Results from the DEC VAX and Telesoft Ada compilers are summarized in the following tables.

Table 1a

Appendix B Weighting Scheme (500 Possible)

Compiler	Applicable Total	Passed		Failed	
		Total	%	Total	%
DEC VAX	464	403	88	56	12
Telesoft	472	317	67	155	33

Table 1b

No Weighting Scheme - Equally Weighted (144 Possible)

Compiler	Applicable Total	Passed		Failed	
		Total	%	Total	%
DEC VAX	138	120	87	18	13
Telesoft	139	94	68	45	32

Table 1a lists the results obtained using the weighting scheme of Appendix B. Table 1b lists the results obtained without using a weighting scheme. The pass and fail percentages listed in these tables are computed using the applicable totals. Since not every test in the test suite was applied to these two compilers, the applicable total is smaller than the total number of possible points.

Table 2
Failure Points Using Appendix B Weighting Scheme

Failure Category	DEC VAX Points	Telesoft Points
Documentation	22	37
Compilation Speed	10	50
Exceeded Capacity	3	17
Code Size Performance	14	21
Code Speed Performance	4	6
Nonstandard (I/O, pragmas, chapter 13)	3	24
Total	56	155

Table 2 contains a breakdown by general type of the tests failed by each of the compilers. The points listed were obtained from the weighting scheme of Appendix B, summarized in Table 1a. The failure categories identified here are not an exhaustive representation of the categories present in the PQAC test suite.

4.2 DEC VAX Results

The DEC VAX compiler achieved a test suite pass percentage of 88% using the weights given in Appendix B. Of the 14 essential requirements, only one test for compilation speed failed.

Table 2 indicates that inferior or missing documentation accounts for more than one third of the points taken off. The compilation speed failure and the non-optimal generation of object code account for most of the remaining points. The results show that there is not a specific area of the compiler that is below the standard.

In this limited two compiler domain, the DEC VAX compiler performed significantly better than did the Telesoft, achieving a rating more than 20% higher. A higher level of production quality is indicated.

4.3 Telesoft Results

The Telesoft compiler achieved a test suite pass percentage of 67% using the weights given in Appendix B. It also failed six of the essential requirements: object code size, object code speed, compiler error rate, field testing, and two for compilation speed. Several library and statement capacity limits were exceeded. Several predefined pragmas and MIL-STD-1815A Chapter 13 constructs such as length clauses were not supported. Inferior or missing documentation was also noted. As shown in Table 2, the DEC VAX compiler performed better than the Telesoft in each of the identified categories.

The Telesoft compiler was more difficult to work with than the DEC VAX. The initial compile of the test suite support software halted with a cryptic error message stating that an internal compiler error had occurred. It was surprising to find that a symbolic debugger was not included with the compiler and that one wasn't purchased separately by Aerospace. Also, a compiler option problem made it impossible to generate a compiler listing. When contacted, Telesoft was able to provide a work around for this problem. Telesoft was previously unaware of two problems for which error reports were submitted.

The PUT procedure from the predefined package TEXT_IO operated differently than expected, necessitating the modification of several tests. Various tests containing large Ada code segments failed after exceeding a CPU time limit or library capacity, including one that failed after using six hours of CPU time.

Telesoft compiler options exist for increasing the working set size and virtual memory used by the compiler. Using these options could potentially improve the compiler's performance for some of the tests such as those for compilation speed. These options were not used, however, since executing many of these tests more than once is cost prohibitive. Additionally, the necessity for such fine tuning demonstrates a lack of ease of use, which is a tacit requirement for production quality.

The difficulties with the Telesoft compiler are captured by the lower pass percentage rating achieved. It is evident from these results that the Telesoft compiler is less mature than the DEC VAX.

5. The *Definition* Analysis

The *Definition* contains an excellent set of requirements specifying desirable attributes that a production quality Ada compiler should possess. Important aspects of an Ada compiler not addressed by other test suites are addressed here. However, certain practical limits are imposed upon such requirements when they are to be translated into tests for verification of the requirement. Creating and applying the PQAC test suite to the DEC VAX and Telesoft Ada compilers have illuminated several of these limitations. A list of specific comments for selected requirements is contained in Appendix E.

5.1 Requirement Review

For a requirement to be of any use, there must be some way that it can actually be tested. Any test method devised for a requirement should be repeatable and consistent with independent testers. A major problem with the *Definition* is that several requirements are difficult, if not impossible, to formulate tests for in the framework of a test suite.

An example of such a requirement is 4.3.2, "The diagnostic message text shall be sufficiently informative to enable the user to analyze the problem without consulting compiler documentation." Although this requirement is certainly a valuable trait for a production quality compiler, as stated it is not quantifiable. In general, phrases in a requirement such as "sufficiently informative" or "adequate" render the requirement toothless. These requirements could be restated in more definite terms.

A more severe problem is with 7.5, "The production quality compiler should exhibit an error rate of no more than 1 verified new error for each 250,000 new lines of Ada compiled. This rate shall decrease over time as the compiler matures." This is obviously a very important requirement. However, verifying that a compiler satisfies this requirement is virtually impossible. It requires the tester to rely on the vendor for adequate compiler history statistics. There is no guarantee that these statistics will be available or accurate. The *Definition* should be tailored in some manner so that these untestable requirements do not invalidate the entire collection. The intended scope of the requirements in the *Definition* may be too broad.

5.2 Deficiencies

Many of the requirements in the *Definition* are also provided with a rationale. This rationale is meant to help explain and interpret the intent of the requirement. It is unclear whether the rationales are to be taken as part of the requirement or whether they are just guidelines.

Information in some of the rationales is indispensable to the correct interpretation of the requirement.

Requirements and rationales that are ambiguous, unclear, or incomplete are given in Appendix E. Requirements such as these need to be more specific. For example, in 4.1.4, what does "implement an option to recover" imply? Several of the requirements in Section 3 that do not have a rationale should have one, since the requirements as stated are unclear. Section 8.4 requires the existence of a Run-Time System Manual. Since a one page or error filled manual is not acceptable, some reference to the quality and content of such a document should be made.

It is desirable to keep the size and scope of the *Definition* compact to facilitate ease of use and analysis. However, additional requirements may be identified that would add to the completeness of the *Definition*. They could include:

- A requirement on the compilation speed of library units having WITH clauses.
- A limit on subprogram declarations in Section 3.2.
- More requirements for size and representation clauses. For example, specifying a size of N bits for some type should cause an M element array of this type to be of size M * N.
- A requirement that the recompilation of a package body should not necessitate the recompilation of any unit dependent on the package.
- Requirements for predefined math libraries, database management systems, and operating system services.

The format of the *Definition* could be modified to facilitate the testing of the requirements. Those requirements that contain multiple requirements, as in 6.12, could be broken up with each subrequirement being assigned its own section number. Requirements that are inseparable, such as 3.4.5 and 3.4.6, could be joined to form one requirement. The requirements could be ordered so that no requirement is dependent on a requirement that follows it. For ease of use, the requirements could contain references to MIL-STD-1815A for those constructs being tested that are described there.

5.3 Summary

The *Ada Compiler Validation Certification* (ACVC) and *Ada Compiler Evaluation Certification* (ACEC) are mainly targeted at measuring adherence to MIL-STD-1815A and run-time performance. Production quality entails more than what either of these benchmark suites provide, although meeting their requirements is an absolute minimum for any Ada implementation.

Ada puts more emphasis on the quality and structure of the programming environment in which it operates than other programming languages. It is crucial that an Ada compiler have available adequate documentation, debuggers, and library management tools. Adherence to the essence of the Ada standard and the support of advanced Ada features for I/O, pragmas, attributes, and MIL-STD-1815A Chapter 13 issues is also critical. The *Definition* addresses issues such as these that are not addressed by other test suites.

The PQAC test suite containing 144 tests has been developed from the requirements in the *Definition*. Most of the requirements (70%) are testable using Ada code, while others rely on performing some manual procedure such as checking for the existence of a User's Manual. For the latter group, the test consists of a description of the manual procedure and criteria to use in evaluating the requirement. Some of the requirements may not be applicable to a given compiler and host/target architecture. For example, the existence of a Software Product Specification is only required when a compiler is being newly developed for a DOD contract.

A capability for assigning different weights to each of the tests was created. The weights assigned may be modified to fit the application. This weighting scheme is used to derive a production quality performance rating between 0% and 100% for an Ada compiler. This rating may then be used to compare Ada compilers, or to flag deficient compilers. A detailed analysis of a deficient compiler can be obtained by reviewing the test data and results. The test suite and weights must be allowed to achieve some level of maturity before any firm conclusions may be drawn from obtained results.

The PQAC test suite was applied to the DEC VAX and Telesoft Ada compilers. Using the weighting scheme developed in this report, the DEC VAX and Telesoft compilers received a production quality performance rating of 88% and 67%, respectively. Although the rating should only be used as an indicator, difficulties encountered with the Telesoft compiler tend to support the obtained results. As more compilers are evaluated using the *Definition*, a standard interpretation of the production quality rating will evolve.

Practical limitations on several of the requirements have been identified. Several missing or incomplete requirements have also been noted. Even with these problems, the *Definition* is, for the most part, a complete and useful set of requirements for measuring a compiler. An Ada community periodic review could be scheduled, allowing new information and improvements to be incorporated into the *Definition*.

Acronyms

ACEC	Ada Compiler Evaluation Certification
ACVC	Ada Compiler Validation Certification
AJPO	Ada Joint Program Office
APSE	Ada Programming Support Environment
CPU	Central Processing Unit
DOD	Department of Defense
ECSP0	Embedded Computer Standardization Program Office
HOL	High Order Language
MIP	Million Instructions Per Second
PIWG	Performance Issues Working Group
PQAC	Production Quality Ada Compiler
SSD	Space Systems Division
VDD	Version Description Document

Appendices

A. Test and Requirement Cross Reference

This appendix provides a correlation between the requirements in the *Definition* and the numbers used by the PQAC test suite. Some of the requirements in the *Definition* have been subdivided in order to keep the tests independent and to make the evaluation of results easier.

The numbers used by the PQAC test suite are listed on the left in bold with their corresponding requirement from the *Definition* stated on the right.

Those requirements that are considered to be fundamentally essential for a production quality compiler and must be met exactly are denoted by "(M)", for "minimal."

SECTION 1 INTRODUCTION

T010100 1.1 Definition of an Ada Source Statement. An Ada source statement shall be defined to mean: a basic declaration, a record component declaration, a simple statement, a compound statement, an entry declaration, terminate alternative, WITH clause, USE clause, generic parameter declaration, proper body or body stub, representation clause, alignment clause, or component clause.

SECTION 2 PERFORMANCE REQUIREMENTS

- T020100** 2.1 Benchmarks Used. All performance requirements of this section shall be met using the programs of the test suite formulated by the Performance Issues Working Group (PIWG) of the SIGAda Users' Committee.
- T020200** 2.2 Definition of Benchmark Test Units. The requirements in this section assume a single compilation unit without any context clauses (WITH clauses) or generic instantiations.
- T020300** 2.3 Definition of Time Used. All speed requirements of this section shall be measured in terms of elapsed (wall-clock) time.
- 2.4 Host System Performance Requirements.
- T020401** 2.4.1 The compiler shall compile a syntactically and semantically correct Ada program of at least 200 Ada source statements at a rate of at least 200 statements per minute (elapsed time), for each 1 MIP of rated processing speed of the specified host computer, while meeting the object code requirements in 2.5.1 and 2.5.2. (M)
- T020402** 2.4.2 The compiler shall compile a syntactically and semantically correct Ada program of at least 200 Ada source statements at a rate of at least 500 statements per minute (elapsed time), for each 1 MIP of rated processing speed of the specified host computer, in the absence of requirements on object code efficiency. (M)
- T020403** 2.4.3 The compiler shall compile a syntactically and semantically correct Ada program of at least 200 Ada source statements at a rate of at least 1000 statements per minute (elapsed time), for each 1 MIP of rated processing speed of the specified host computer, with no requirement to generate object code.
- 2.5 Target System Performance Requirements.
- T020501** 2.5.1 The compiler shall produce an object code program that requires no more than 30% additional target computer memory space over an equivalent program written in assembly language. (M)
- T020502** 2.5.2 The compiler shall produce an object code program that requires no more than 15% additional execution time over an equivalent program written in assembly language. (M)

SECTION 3 COMPILER CAPACITY REQUIREMENTS

- 3.1 Program Limitations. The compiler shall provide the following minimum capacities for each of the Ada program elements listed when provided with sufficient virtual storage:

<i>T030101</i>	library units in a program library	2048
<i>T030102</i>	compilation units in a program	1024
<i>T030103</i>	Ada source statements in a program	2,500,000
<i>T030104</i>	maximum size (in words) of a program	2,500,000
<i>T030105</i>	ELABORATE pragmas	512
<i>T030106</i>	width of source line (and length of identifier)	120

- 3.2 Compilation Unit Limitations. The compiler shall provide the following minimum capacities for each of the compilation unit elements listed when provided with sufficient virtual storage:

<i>T030201</i>	library units in a single context clause	16
<i>T030202</i>	library units WITHed by a compilation unit	256
<i>T030203</i>	external names	4096
<i>T030204</i>	Ada source statements in a compilation unit	4096
<i>T030205</i>	identifiers (including those in WITHed units)	4096
<i>T030206</i>	declarations (total) in a compilation unit	4096
<i>T030207</i>	type declarations	1024
<i>T030208</i>	subtype declarations of a single type	1024
<i>T030209</i>	literals in a compilation unit	1024

- 3.3 Program Unit Limitations. The compiler shall provide the following minimum capacities for each of the program unit (subprogram, package, task, or generic unit body) elements listed when provided with sufficient virtual storage:

<i>T030301</i>	depth of nesting of program units	64
<i>T030302</i>	depth of nesting of blocks	64
<i>T030303</i>	depth of nesting of case statements	64
<i>T030304</i>	depth of nesting of loop statements	64
<i>T030305</i>	depth of nesting of if statements	256
<i>T030306</i>	elsif alternatives	256
<i>T030307</i>	exception declarations in a frame	256
<i>T030308</i>	exception handlers in a frame	256
<i>T030309</i>	declarations in a declarative part	1024
<i>T030310</i>	identifiers in a declarative part	1024
<i>T030311</i>	frames an exception can propagate through	unlimited

3.4 Task Limitations. The compiler shall provide the following minimum capacities for each of the task elements listed when provided with sufficient virtual storage:

T030401	values in subtype SYSTEM.PRIORITY	16
T030402	simultaneously active tasks in a program	512
T030403	accept statements in a task	64
T030404	entry declarations in a task	64
T030405	formal parameters in an entry declaration	64
T030406	formal parameters in an accept statement	64
T030407	delay statements in a task	64
T030408	alternatives in a select statement	64

3.5 Subprogram Limitations. The compiler shall provide the following minimum capacities for each of the subprogram elements listed when provided with sufficient virtual storage:

T030501	formal parameters	64
T030502	levels in a call chain	unlimited

3.6 Package Limitations. The compiler shall provide the following minimum capacities for each of the package elements listed when provided with sufficient virtual storage:

T030601	visible declarations	1024
T030602	private declarations	1024

3.7 Statement Limitations. The compiler shall provide the following minimum capacities for each of the statement elements listed when provided with sufficient virtual storage:

T030701	declarations in a block	1024
T030702	enumeration literals in a single type	512
T030703	dimensions in an array	32
T030704	total elements in an array	65535
T030705	components in a record type	256
T030706	discriminants in a record type	64
T030707	variant parts in a record type	64
T030708	size of any object in bits	65535
T030709	characters in a value of type STRING	65535

3.8 Expression Limitations. The compiler shall provide the following minimum capacities for each of the expression elements listed when provided with sufficient virtual storage:

T030801	operators in an expression	128
T030802	function calls in an expression	128
T030803	primaries in an expression	128
T030804	depth of parentheses nesting	64

SECTION 4 USER INTERFACE REQUIREMENTS

4.1 User Inputs.

- T040101** 4.1.1 The compiler shall be invocable from either a batch file command or an interactive command.
- T040102** 4.1.2 The compiler shall be sharable (re-entrant) by multiple users, if the host operating system supports multiple users.
- T040103** 4.1.3 The compiler shall implement options to perform the same function as pragmas *SUPPRESS* and *OPTIMIZE*.
- T040104** 4.1.4 The compiler shall implement an option to recover from non-fatal errors as defined in 4.3.3. The recovery action taken shall be identified.
- T040105** 4.1.5 The compiler shall implement an option to disable the generation of diagnostic messages of a specified severity level.
- T040106** 4.1.6 The compiler shall implement an option to select or suspend the generation of object code and/or assembly code.

4.2 Compiler Listings.

- T040201** 4.2.1 The compiler shall be able to produce at the option of the user a compilation listing showing the source code with line numbers.
- T040202** 4.2.2 The compiler shall be able to produce at the option of the user a list of diagnostic messages either at the position in the source code where the condition occurred, and/or at the end of the compilation listing, even if the compilation terminates abnormally.
- T040203** 4.2.3 The compiler shall be able to produce at the option of the user an assembly or pseudo-assembly output listing.
- T040204** 4.2.4 The compiler shall be able to produce at the option of the user an assembly or pseudo-assembly output listing with embedded Ada source statements adjacent to the assembly code they generated.
- T040205** 4.2.5 The compiler shall be able to produce at the option of the user a cross reference (set/use) listing.
- T040206** 4.2.6 The compiler shall be able to produce at the option of the user a map of relative addresses of variables and constants.

T040207 4.2.7 For each compilation, the compiler shall be able to produce at the option of the user a statistics summary listing with the following information:

- a. Number of statements
- b. Number of source lines
- c. Compile time per program module (CPU time)
- d. Total compile time (CPU and elapsed time)
- e. Total number of instructions generated
- f. Total number of data words generated
- g. Total size of object module generated

T040208 4.2.8 All listings shall include the following header information on every page:

- a. Date and time of compilation
- b. Compilation unit name
- c. Type of listing
- d. Page number within total listing
- e. User identification

T040209 4.2.9 All listings shall have the following additional information within the listing:

- a. Compiler name, version number, release date
- b. Host and target computer configurations
- c. Specified and default control options
- d. Source file name
- e. Object file name

4.3 Diagnostic Messages.

T040301 4.3.1 Each diagnostic message shall contain the message text, a reference number for additional information in the compiler documentation, and a severity level.

T040302 4.3.2 The diagnostic message text shall be sufficiently informative to enable the user to analyze the problem without consulting compiler documentation.

T040303 4.3.3

The severity levels of diagnostic messages shall include the following error classes:

- a. Note: Information to the user; the compilation process continues and the object program is not affected.
- b. Warning: Information about the validity of the program. The source program is well-defined and semantically correct; the object program may not behave as intended.
- c. Error: An illegal syntactic or semantic construct with a well-defined recovery action. Compilation continues and the object program contains code for the illegal construct; the object program may behave meaninglessly at run-time.
- d. Serious Error: Illegal construct with no well-defined recovery action. Syntax analysis continues but no object program is generated.
- e. Fatal Error: Illegal construct with no reasonable syntactic recovery action. Compilation terminates and no outputs other than the source listing and diagnostic messages are produced.

T040304 4.3.4

The compiler shall issue a diagnostic message to indicate any capacity requirements that have been exceeded.

T040305 4.3.5

The compiler shall not abort regardless of the type or number of errors encountered.

SECTION 5

EXTERNAL TOOLS INTERFACE REQUIREMENTS

5.1 Listing Tools.

- T050101** 5.1.1 The compiler and/or external tool shall be able to produce a source listing with indentations to show control constructs.
- T050102** 5.1.2 The compiler, linker/loader, and/or external tool shall be able to produce an absolute assembly code listing.
- T050103** 5.1.3 The compiler and/or library manager shall be able to produce at the option of the user a dependency listing showing which library units are WITHed by other units.
- T050104** 5.1.4 The compiler and/or library manager shall have the capability of listing all out-of-date (obsolete) library units with the option of selectively recompiling such units before linking.

5.2 Linker/Loader.

- T050201** 5.2.1 The compiler and/or linker/loader shall include in the load module only those subprograms that are actually referenced by the object program.
- T050202** 5.2.2 The compiler and/or linker/loader shall include in the load module only those run-time system modules that are referenced by the object program.
- T050203** 5.2.3 The compiler and/or linker/loader shall support the partial linking of object modules as specified by the user.
- T050204** 5.2.4 The compiler and/or linker/loader shall support the linking of designated object modules without including them in the load module.
- T050300** 5.3 Symbolic Debugger. The compiler shall be able to produce object code files and other types of data necessary to debug those files with an available source-level (symbolic) debugger.

SECTION 6

ADA LANGUAGE REQUIREMENTS

- T060100** 6.1 General. The compiler shall eliminate statements or subprograms that will never be executed (dead code) because their execution depends on a condition known to be false at compilation time.
- 6.2 Character Sets.
- T060201** 6.2.1 The compiler shall allow the Ada program text to contain any of the 95 graphic characters and 5 form effectors of the ISO 7-bit character set (ISO Standard 646) to the extent supported by the host computer.
- T060202** 6.2.2 The predefined packages TEXT_IO, DIRECT_IO, and SEQUENTIAL_IO shall support input and output of data containing any of the 128 ASCII character literals of the predefined type STANDARD.CHARACTER.
- T060203** 6.2.3 The compiler shall allow comments and values of the predefined type STRING to contain any of the 128 ASCII characters contained in the predefined type STANDARD.CHARACTER.
- 6.3 Data Representation.
- T060301** 6.3.1 The compiler shall provide predefined types in package STANDARD for all the integer and floating-point types provided by the target computer.
- T060302** 6.3.2 The compiler shall support universal integer calculations requiring up to 64 bits of accuracy.
- T060303** 6.3.3 The components of array types with BOOLEAN components named in a pragma PACK shall be stored in contiguous memory bits, i.e., each component shall occupy only one bit of storage.
- T060304** 6.3.4 The compiler shall support address clauses.
- T060305** 6.3.5 The compiler shall support length clauses, enumeration representation clauses, and record representation clauses.
- T060306** 6.3.6 The range of integer code values allowed in an enumeration representation clause shall be MIN_INT to MAX_INT.
- T060307** 6.3.7 The compiler shall allow non-contiguous integer code values in an enumeration representation clause.
- T060308** 6.3.8 The compiler shall support the SIZE attribute designator for enumeration types named in a length clause.

- T060309** 6.3.9 The compiler shall support the *SMALL* attribute designator for fixed point types.
- T060310** 6.3.10 Memory space for the creation of objects designated by an access type shall not be allocated until allocators (new statements) for that type are executed.

6.4 Subprograms.

- T060401** 6.4.1 The compiler shall expand inline any subprogram or generic subprogram instantiation that is named in a pragma *INLINE* and that meets the criteria of 6.4.2.
- T060402** 6.4.2 A subprogram or generic subprogram instantiation is a candidate for inline expansion if it meets the following criteria:
- a. Its body is declared in either the current unit or the compilation library.
 - b. Its parameters or result type (for functions) are not task types, composite types with task type components, unconstrained array types, or unconstrained types with discriminants.
 - c. It does not contain another subprogram body, package body, body stub, generic declaration, generic instantiation, exception declaration, or access type declaration.
 - d. It does not contain declarations that imply the creation of dependent tasks.
 - e. It does not contain any subprogram calls that result in direct or indirect recursion.
- T060403** 6.4.3 The compiler shall expand inline any subprogram that meets the requirements in 6.4.2 and that is called only once.

- T060404** 6.4.4 The compiler shall provide the capability for main subprograms to return a value to the target computer run-time system indicating the completion status of the program.

6.5 Tasking.

- T060501** 6.5.1 The compiler shall provide a capability for handling target computer hardware or operating system interrupts as calls to Ada task entries.
- T060502** 6.5.2 The execution-time overhead to perform a context switch or to terminate or abort a task shall be no more than that required to call or return from a subprogram.

- T060503** 6.5.3 The ordering of select alternatives in a selective wait statement shall not impact the execution speed of the program.
- T060504** 6.5.4 The compiler shall dispatch the execution of ready tasks in a manner that will give each task an equal share of the processing resources consistent with any PRIORITY pragmas.
- T060505** 6.5.5 Tasks that are blocked, completed, terminated, or not activated shall not impact the performance of the active tasks.
- T060506** 6.5.6 The value of DURATION'DELTA shall not be greater than 1 millisecond.

6.6 Exceptions.

- T060601** 6.6.1 An exception shall not impact execution speed until it is raised.
- T060602** 6.6.2 The compiler shall provide the pragma SUPPRESS or an equivalent capability to permit suppression of all predefined run-time checks in a designated compilation unit.
- T060603** 6.6.3 The compiler shall issue a warning message to indicate static expressions that will always raise a constraint exception at run-time.

6.7 Generics.

- T060701** 6.7.1 The compiler shall share code between multiple instantiations of generic units that do not differ in their underlying machine representation.
- T060702** 6.7.2 The compiler shall allow generic specifications and bodies to be compiled in completely separate compilations.
- T060703** 6.7.3 The compiler shall allow subunits of a generic unit to be separately compiled.

6.8 Interface with Other Languages.

- T060801** 6.8.1 The compiler shall provide the pragma INTERFACE to allow importing of assembly language programs already assembled into the object code format of the target computer. The machine language interface for procedure and function parameters and function result types shall be documented.
- T060802** 6.8.2 The compiler shall provide the pragma INTERFACE, or an equivalent mechanism, to allow incorporation of subprogram bodies compiled from the standard system or application language of the target computer.

T060900 6.9 Unchecked Programming. The generic library subprograms UNCHECKED_DEALLOCATION and UNCHECKED_CONVERSION shall be implemented with no restrictions except that both objects in an unchecked conversion may be required to be of the same size.

6.10 Input/Output.

T061001 6.10.1 An implementation shall provide packages to allow input and output of FORTRAN-formatted text files for each target computer that supports text input/output.

T061002 6.10.2 Package SEQUENTIAL_IO and package DIRECT_IO shall be able to be instantiated with unconstrained array types or with unconstrained record types which have discriminants without default values.

T061003 6.10.3 The compiler shall allow more than one internal file to be associated with each external file for DIRECT_IO and SEQUENTIAL_IO for both reading and writing.

T061004 6.10.4 The compiler shall allow an external file associated with more than one internal file to be deleted.

6.11 System Information.

T061101 6.11.1 The named numbers defined in package SYSTEM shall not limit or restrict the inherent capabilities of the target computer hardware or operating system.

T061102 6.11.2 The enumeration type NAME defined in PACKAGE SYSTEM shall have values for all target computers for which the compiler generates code.

6.12 Pragmas. An implementation shall provide the predefined pragmas

T061201	CONTROLLED,
T061202	ELABORATE,
T061203	LIST,
T061204	MEMORY_SIZE,
T061205	OPTIMIZE,
T061206	PAGE,
T061207	STORAGE_UNIT,
T061208	and SYSTEM_NAME.

SECTION 7

QUALITY ASSURANCE AND RELIABILITY REQUIREMENTS

- T070100 7.1** Validation. The compiler shall be validated by an Ada Validation Facility established and operated under the direction of the DOD Ada Joint Program Office in all configurations necessary to meet the requirements of this document. (M)
- T070200 7.2** Field Testing. The compiler shall be subjected to a minimum of 20 site-months of independent evaluation and usage in a realistic production work environment before release for production use. (M)
- T070300 7.3** Maintenance. Provisions for on-going problem correction of the compiler shall be provided.
- T070400 7.4** Configuration Management. The maintaining organization shall provide configuration management for the compiler, including maintenance of an up-to-date data base of compiler errors showing the nature and status of each error.
- T070500 7.5** Error Rate. The production quality compiler should exhibit an error rate of no more than 1 verified new error for each 250,000 new lines of Ada compiled. This rate shall decrease over time as the compiler matures. (M)

SECTION 8 DOCUMENTATION REQUIREMENTS

- T080100 8.1** Validation Summary Report. The vendor shall provide a copy of the most recent version of the official validation summary report prepared by the Ada Validation Organization that validated the compiler. (M) This report shall include both CPU and elapsed times required to run the ACVC tests.
- T080200 8.2** Ada Language Reference Manual (ARM). The compiler vendor shall supply a copy of the Ada Language Reference Manual (ARM) (ANSI/MIL-STD 1815A) that includes implementation-specific details of the compiler where applicable. (M)
- T080300 8.3** User's Manual. The vendor shall provide a User's Manual that describes how to use the compiler to develop Ada applications programs, including information on how to run the compiler. It shall include all system-dependent forms implemented in the compiler (i.e., machine-specific functions), methods of selecting debug aids, compiler options and parameters, and a complete list of error and warning messages provided by the compiler, with a description of each. Message descriptions shall reference the relevant section of the ARM. The manual shall include examples of the commands used to invoke the compiler and linker/loader system with various combinations of compiler and linker options, respectively. (M)
- T080400 8.4** Run-Time System Manual. The vendor shall provide a Run-time System Manual for each target computer. (M)
- T080500 8.5** Version Description Document. The vendor shall provide a Version Description Document for each compiler configuration. (M)
- T080600 8.6** Installation Manual. The vendor shall provide a detailed Installation Manual and all the necessary software materials for installing each host configuration of the Ada compiler, including several sample Ada programs with correct output. (M)
- T080700 8.7** Maintenance Manual. The vendor shall provide a Maintenance Manual which presents the methods to be used in the general maintenance of all parts of the compiler. All major data structures, such as the symbol table and the intermediate language, shall be fully described. All debugging aids that have been inserted into the compiler shall be described and their use fully stated. If the compiler has a special "maintenance mode" of operation to assist in pinpointing errors, this shall be fully described.
- T080800 8.8** Software Product Specification. The vendor shall provide a Software Product Specification for the compiler in accordance with DOD-STD-2167A and Data Item Description DI-MCCR-80029A. (M)

B. PQAC Test Weights

This appendix contains the weights assigned for each individual requirement identified in Appendix A. The method used for awarding points to a partial pass is also given here.

Example: *T020401* tests for a compilation speed of 200 lines/minute/MIP. The instruction for this test is to grade from 1 - 10 for speeds from 100 - 200. If a compiler achieves a speed of 190 lines/minute/MIP, then this instruction would award the test 9 out of 10 points. Similarly, a speed of 150 would rate 5 out of 10 points. A speed of less than 100 would not receive any points.

Partial credit is also allowed for some of the tests because multiple cases are being tested at the same time. These tests are also identified in the table. The requirements that consist solely of a definition and have no corresponding test are not assigned a weight.

Num	Test	Weight	Special Grading Instructions
1.	<i>T010100</i>	N/A	Definition
2.	<i>T020100</i>	N/A	Definition
3.	<i>T020200</i>	N/A	Definition
4.	<i>T020300</i>	N/A	Definition
5.	<i>T020401</i>	10	Grade from 1 - 10 for speeds from 100 - 200
6.	<i>T020402</i>	10	Grade from 1 - 10 for speeds from 250 - 500
7.	<i>T020403</i>	10	Grade from 1 - 10 for speeds from 500 - 1000
8.	<i>T020501</i>	10	Grade from 10 down to 1 for percent from 30 - 60
9.	<i>T020502</i>	10	Grade from 10 down to 1 for percent from 15 - 30
10.	<i>T030101</i>	2	Give 1 point if 1024 allowed
11.	<i>T030102</i>	2	Give 1 point if 512 allowed
12.	<i>T030103</i>	2	Give 1 point if 1,000,000 allowed
13.	<i>T030104</i>	2	Give 1 point if 1,000,000 allowed
14.	<i>T030105</i>	1	
15.	<i>T030106</i>	1	
16.	<i>T030201</i>	1	
17.	<i>T030202</i>	2	Give 1 point if 128 allowed
18.	<i>T030203</i>	2	Give 1 point if 2048 allowed
19.	<i>T030204</i>	2	Give 1 point if 2048 allowed
20.	<i>T030205</i>	2	Give 1 point if 2048 allowed

PQAC Test Weights

Num	Test	Weight	Special Grading Instructions
21.	T030206	2	Give 1 point if 2048 allowed
22.	T030207	1	
23.	T030208	1	
24.	T030209	1	
25.	T030301	1	
26.	T030302	1	
27.	T030303	1	
28.	T030304	1	
29.	T030305	1	
30.	T030306	1	
31.	T030307	1	
32.	T030308	1	
33.	T030309	1	
34.	T030310	1	
35.	T030311	2	
36.	T030401	1	
37.	T030402	2	Give 1 point if 256 allowed
38.	T030403	1	
39.	T030404	1	
40.	T030405	1	
41.	T030406	1	
42.	T030407	1	
43.	T030408	1	
44.	T030501	2	
45.	T030502	2	
46.	T030601	2	Give 1 point if 512 allowed
47.	T030602	2	Give 1 point if 512 allowed
48.	T030701	2	Give 1 point if 512 allowed
49.	T030702	1	
50.	T030703	1	
51.	T030704	2	
52.	T030705	1	
53.	T030706	1	
54.	T030707	1	
55.	T030708	2	
56.	T030709	2	
57.	T030801	1	
58.	T030802	1	
59.	T030803	1	
60.	T030804	2	
61.	T040101	10	
62.	T040102	10	
63.	T040103	4	Give 2 points for each option
64.	T040104	4	Give 2 points for identifying the recovery
65.	T040105	2	

PQAC Test Weights

Num	Test	Weight	Special Grading Instructions
66.	T040106	2	
67.	T040201	10	
68.	T040202	10	
69.	T040203	5	
70.	T040204	5	
71.	T040205	5	
72.	T040206	5	
73.	T040207	4	Give 1 - 4 points for 4 - 7 statistics printed
74.	T040208	1	
75.	T040209	1	
76.	T040301	2	Give 1 point if 2 of these items are present
77.	T040302	2	
78.	T040303	2	Give 1 point if 3 error classes identified
79.	T040304	2	
80.	T040305	2	
81.	T050101	2	
82.	T050102	2	
83.	T050103	6	
84.	T050104	6	
85.	T050201	5	
86.	T050202	5	
87.	T050203	4	
88.	T050204	4	
89.	T050300	10	
90.	T060100	4	
91.	T060201	4	
92.	T060202	3	Give 1 point for each package
93.	T060203	4	
94.	T060301	4	
95.	T060302	4	
96.	T060303	4	
97.	T060304	4	
98.	T060305	6	Give 2 points for each clause type
99.	T060306	2	
100.	T060307	4	
101.	T060308	4	
102.	T060309	4	
103.	T060310	4	
104.	T060401	8	
105.	T060402	N/A	Definition
106.	T060403	2	
107.	T060404	4	
108.	T060501	4	
109.	T060502	2	

PQAC Test Weights

Num	Test	Weight	Special Grading Instructions
110.	T060503	1	
111.	T060504	4	
112.	T060505	2	
113.	T060506	2	
114.	T060601	2	
115.	T060602	2	
116.	T060603	2	
117.	T060701	2	
118.	T060702	2	
119.	T060703	2	
120.	T060801	2	
121.	T060802	2	
122.	T060900	4	Give 2 points for each
123.	T061001	2	
124.	T061002	4	Give 1 point for each configuration
125.	T061003	4	Give 1 point for each configuration
126.	T061004	2	
127.	T061101	4	
128.	T061102	2	
129.	T061201	2	
130.	T061202	2	
131.	T061203	1	
132.	T061204	2	
133.	T061205	1	
134.	T061206	1	
135.	T061207	1	
136.	T061208	1	
137.	T070100	10	
138.	T070200	8	
139.	T070300	10	
140.	T070400	8	
141.	T070500	8	
142.	T080100	8	
143.	T080200	10	
144.	T080300	10	
145.	T080400	10	
146.	T080500	8	
147.	T080600	8	
148.	T080700	8	
149.	T080800	8	
	Total	500	

C. DEC VAX Ada Result Summary

This appendix contains a summary of the results of applying the PQAC test suite to the DEC VAX V1.4 Ada Compiler. Appendix A contains a cross reference for the test numbers used in the tables below. Appendix B lists the weights assigned to each test from the PQAC test suite.

Points are distributed for each test between N/A, Pass, and Fail. The total weight for each test may be obtained by adding these three values. It is possible for a test to receive partial pass credit.

A comment describing exceptional conditions or explaining a failure is included for some of the tests. There are 149 identified tests.

Num	Test	N/A	Pass	Fail	Comments:
1.	T010100	0			Definition
2.	T020100	0			Definition
3.	T020200	0			Definition
4.	T020300	0			Definition
5.	T020401		10		
6.	T020402			10	Achieved only 200 lines/minute/MIP
7.	T020403		10		
8.	T020501		10		
9.	T020502		10		
10.	T030101		2		
11.	T030102		1	1	Limit = 1000
12.	T030103		2		
13.	T030104		2		
14.	T030105		1		
15.	T030106		1		
16.	T030201		1		
17.	T030202		2		
18.	T030203		2		
19.	T030204		2		
20.	T030205		2		

DEC VAX Ada Result Summary

Num	Test	N/A	Pass	Fail	Comments:
21.	T030206		2		
22.	T030207		1		
23.	T030208		1		
24.	T030209		1		
25.	T030301		1		
26.	T030302		1		
27.	T030303		1		
28.	T030304		1		
29.	T030305		1		
30.	T030306		1		
31.	T030307		1		
32.	T030308		1		
33.	T030309		1		
34.	T030310		1		
35.	T030311			2	Limit = 65535
36.	T030401		1		
37.	T030402		2		
38.	T030403		1		
39.	T030404		1		
40.	T030405		1		Limit of 32 on unconstrained record types
41.	T030406		1		
42.	T030407		1		
43.	T030408		1		
44.	T030501		2		Limit of 32 on unconstrained record types
45.	T030502		2		
46.	T030601		2		
47.	T030602		2		
48.	T030701		2		
49.	T030702		1		
50.	T030703		1		
51.	T030704		2		
52.	T030705		1		
53.	T030706		1		
54.	T030707		1		
55.	T030708		2		
56.	T030709		2		
57.	T030801		1		
58.	T030802		1		
59.	T030803		1		
60.	T030804		2		
61.	T040101		10		
62.	T040102		10		
63.	T040103		4		
64.	T040104		4		
65.	T040105		2		

DEC VAX Ada Result Summary

Num	Test	N/A	Pass	Fail	Comments:
66.	T040106		2		
67.	T040201		10		
68.	T040202		10		
69.	T040203		5		
70.	T040204		5		
71.	T040205		5		
72.	T040206			5	External tool required
73.	T040207		1	3	Information missing
74.	T040208		1		
75.	T040209			1	Information missing
76.	T040301		2		
77.	T040302		2		
78.	T040303		1	1	One class is missing
79.	T040304		2		
80.	T040305		2		
81.	T050101			2	None available
82.	T050102		2		
83.	T050103		6		
84.	T050104		6		
85.	T050201			5	Extra code included
86.	T050202			5	Extra code included
87.	T050203	4			Application (target/host) specific
88.	T050204	4			Application (target/host) specific
89.	T050300		10		
90.	T060100		4		
91.	T060201		4		
92.	T060202		3		
93.	T060203		4		
94.	T060301		4		
95.	T060302		4		
96.	T060303		4		
97.	T060304		4		
98.	T060305		6		
99.	T060306		2		
100.	T060307		4		
101.	T060308		4		
102.	T060309		4		
103.	T060310		4		
104.	T060401		8		
105.	T060402	0			Definition
106.	T060403			2	Code not expanded inline
107.	T060404		4		
108.	T060501	4			Not applicable to VAX VMS
109.	T060502			2	Order of magnitude time difference

DEC VAX Ada Result Summary

Num	Test	N/A	Pass	Fail	Comments:
110.	T060503		1		
111.	T060504		4		
112.	T060505		2		
113.	T060506		2		
114.	T060601			2	20% time difference
115.	T060602		2		
116.	T060603		2		
117.	T060701			2	Code not shared
118.	T060702		2		
119.	T060703		2		
120.	T060801		2		
121.	T060802		2		
122.	T060900		4		
123.	T061001			2	None available
124.	T061002		4		
125.	T061003		2	2	Failed for writing
126.	T061004		2		
127.	T061101		4		
128.	T061102		2		
129.	T061201		2		
130.	T061202		2		
131.	T061203		1		
132.	T061204		2		
133.	T061205		1		
134.	T061206		1		
135.	T061207			1	Present, but accepts only 1 value
136.	T061208		1		
137.	T070100		10		VAX Ada V1.3 Expires 12-17-87
138.	T070200		8		
139.	T070300		10		
140.	T070400			8	Not provided
141.	T070500	8			Statistics not available
142.	T080100		8		Validation reports are public
143.	T080200		10		
144.	T080300		10		
145.	T080400		10		
146.	T080500		8		
147.	T080600		8		
148.	T080700	8			No internal maintenance
149.	T080800	8			Not under contract
Totals		36	408	56	

D. Telesoft VAX Ada Result Summary

This appendix contains a summary of the results of applying the PQAC test suite to the Telesoft TeleGen2 VAX Ada Compiler. Appendix A contains a cross reference for the test numbers used in the tables below. Appendix B lists the weights assigned to each test from the PQAC test suite.

Points are distributed for each test between N/A, Pass, and Fail. The total weight for each test may be obtained by adding these three values. It is possible for a test to receive partial pass credit.

A comment describing exceptional conditions or explaining a failure is included for some of the tests. There are 149 identified tests.

Num	Test	N/A	Pass	Fail	Comments:
1.	T010100	0			Definition
2.	T020100	0			Definition
3.	T020200	0			Definition
4.	T020300	0			Definition
5.	T020401			10	Requirement precondition not met
6.	T020402			10	Achieved less than 100 lines/minute/MIP
7.	T020403			10	Achieved less than 150 lines/minute/MIP
8.	T020501			10	Required more than 50% additional space
9.	T020502			10	Required more than 100% additional time
10.	T030101			2	Library limits exceeded
11.	T030102			2	Library limits exceeded
12.	T030103			2	Library limits exceeded
13.	T030104			2	Failed due to previous errors
14.	T030105		1		
15.	T030106		1		
16.	T030201		1		
17.	T030202		2		
18.	T030203		2		
19.	T030204		2		
20.	T030205		2		

Telesoft VAX Ada Result Summary

Num	Test	N/A	Pass	Fail	Comments:
21.	T030206		2		
22.	T030207		1		
23.	T030208		1		
24.	T030209		1		
25.	T030301		1		
26.	T030302		1		
27.	T030303		1		
28.	T030304		1		
29.	T030305			1	Parse stack overflow
30.	T030306		1		
31.	T030307			1	Exception limit exceeded
32.	T030308		1		
33.	T030309		1		
34.	T030310		1		
35.	T030311		2		
36.	T030401		1		
37.	T030402		2		
38.	T030403		1		
39.	T030404		1		
40.	T030405		1		
41.	T030406		1		
42.	T030407		1		
43.	T030408		1		
44.	T030501		2		
45.	T030502		2		
46.	T030601		2		
47.	T030602		2		
48.	T030701		2		
49.	T030702		1		
50.	T030703		1		
51.	T030704			2	Capacity exceeded
52.	T030705		1		
53.	T030706		1		
54.	T030707			1	Capacity exceeded
55.	T030708			2	Capacity exceeded
56.	T030709			2	Capacity exceeded
57.	T030801		1		
58.	T030802		1		
59.	T030803		1		
60.	T030804		2		
61.	T040101		10		
62.	T040102		10		
63.	T040103		4		
64.	T040104		4		
65.	T040105			2	No option available

Telesoft VAX Ada Result Summary

Num	Test	N/A	Pass	Fail	Comments:
66.	T040106		2		
67.	T040201		10		
68.	T040202		10		
69.	T040203		5		
70.	T040204		5		
71.	T040205		5		
72.	T040206			5	No option available
73.	T040207			4	Information missing
74.	T040208			1	Information missing
75.	T040209			1	Information missing
76.	T040301		1	1	Information missing
77.	T040302			2	Information missing
78.	T040303		1	1	These classes not identified
79.	T040304			2	Message not given
80.	T040305		2		
81.	T050101		2		
82.	T050102		2		
83.	T050103		6		
84.	T050104		6		
85.	T050201			5	Extra code included
86.	T050202		5		
87.	T050203	4			Application (target/host) specific
88.	T050204	4			Application (target/host) specific
89.	T050300		10		
90.	T060100			4	Dead code not eliminated
91.	T060201		4		
92.	T060202		3		
93.	T060203		4		
94.	T060301			4	Not all types provided
95.	T060302		4		
96.	T060303		4		
97.	T060304		4		
98.	T060305		4	2	Length clauses not supported
99.	T060306		2		
100.	T060307		4		
101.	T060308			4	Length clauses not supported
102.	T060309		4		
103.	T060310		4		
104.	T060401			8	Code not expanded inline
105.	T060402	0			Definition
106.	T060403			2	Code not expanded inline
107.	T060404		4		
108.	T060501	4			Not applicable to VAX VMS
109.	T060502			2	Order of magnitude time difference

Telesoft VAX Ada Result Summary

Num	Test	N/A	Pass	Fail	Comments:
110.	T060503		1		
111.	T060504		4		
112.	T060505			2	Time with tasks aborted excessive
113.	T060506		2		
114.	T060601			2	15% time difference
115.	T060602			2	No pragma available
116.	T060603		2		
117.	T060701			2	Code not shared
118.	T060702		2		
119.	T060703		2		
120.	T060801		2		
121.	T060802		2		
122.	T060900		4		
123.	T061001			2	None available
124.	T061002			4	Not allowed
125.	T061003		2	2	Failed for writing
126.	T061004			2	Deletion not allowed
127.	T061101		4		
128.	T061102		2		
129.	T061201		2		
130.	T061202		2		
131.	T061203		1		
132.	T061204			2	Pragma has no effect
133.	T061205		1		
134.	T061206			1	Pragma has no effect
135.	T061207			1	Pragma has no effect
136.	T061208		1		Only one value in SYSTEM.NAME
137.	T070100		10		
138.	T070200			8	No evaluation of VAX release
139.	T070300		10		
140.	T070400		8		
141.	T070500			8	Compiler errors exceed limit
142.	T080100		8		Validation reports are public
143.	T080200		10		
144.	T080300		10		
145.	T080400		10		
146.	T080500		8		
147.	T080600		8		
148.	T080700	8			No internal maintenance
149.	T080800	8			Not under contract
Totals		28	317	155	

E. Additional Comments

This appendix contains a list of comments for selected requirements in the *Definition*.

T020200 2.2 The PIWG subprogram modules (required by 2.1) without WITH clauses (required by 2.2) have no executable parts. This leaves an absence of code with which to test 2.5.1 and 2.5.2. For the purpose of this report only, a representative algorithm was chosen and modified so that it satisfied all of the requirements of this section except for 2.1.

An executable Ada object would most certainly need WITH clauses or it would be trivial. If you do not test for compilation speed for compilation units with WITH clauses here, then where do you test it?

T020401 2.4.1 Since the requirements in 2.5.1 and 2.5.2 cannot be tested concurrently, this requirement must be satisfied for both cases requiring two tests. A note to this effect should be included in the statement of the requirement.

Absolute compilation speed is not only affected by CPU speed but by disk speed. This measure should not be made absolute but against other compilers performing functionally equivalent compilations. Compilation rate is obviously affected by the type of language features tested (e.g. comment %, blankline %, keyword %, generics, tasking, WITH'ed packages, etc.). The type of language features involved should be addressed by the requirement.

The minimum value stated for this and several other requirements is suspect. What if a compiler achieves a speed of 500 lines/minute/MIP? Or, in the case of the Amdal, 6666 lines/minute/MIP (80K lines total)? The requirements should be stated to reward such performance. Such requirement values should be allowed to change over time as Ada compilers become more efficient.

T020501 2.5.1 Due to the difficulty of procuring assembly language code for every machine to be tested, in addition to possibly being inappropriate, the timing comparisons have been made in terms of equivalent optimized FORTRAN code. It may be even more appropriate to make comparisons to compilers for languages of comparable complexity (Jovial, PL 1), since much more is being accomplished by the Ada compiler in terms of type checking and task scheduling than the assembly code would be.

It would be advantageous to use existing standards (such as the Whetstone and Dhrystone benchmarks) for compilation/execution speed comparisons of Ada compilers against production quality compilers of other languages. These standards are accepted by the community and performance characteristics using them may already be available. However, the requirement of 2.2 (no WITH clauses) would make existing standards difficult (requiring modification) if not impossible to use.

T020502 2.5.2 See previous comment.

T030103 3.1.3 This requirement needs to address what kind of statements to use in the compilation. A program can be written containing 2,500,000 lines that will be accepted by almost every compiler (2,499,999 null statements). A program can also be written containing 2,500,000 lines that will be rejected by every compiler (multiple advanced Ada features). In general, it is difficult to construct a large program such that if the compiler rejects it, it is guaranteed to be a compiler problem rather than a problem with disk storage, memory size, computing time, etc.

The requirements in 3.1.2 (1024 compilation units in a program) and 3.2.4 (4096 Ada source statements in a compilation unit) already give a combined implicit requirement for 4,194,304 Ada source statements in a program.

In addition, this and several other requirements of this section are very expensive to test. This requirement used over 5 hours of CPU time for the DEC VAX compiler.

T030104 3.1.4 This requirement is dependent on the compiler application and configuration. If the target computer is a 1750A with 8K of RAM, this requirement would not be appropriate.

The word size is not defined for this requirement. Does this mean target or host word size? This may make it difficult to compare the results of compilers based on different word sizes if such a comparison need ever be made. This requirement could be expressed in terms of bit size.

T030105 3.1.5 The requirement for the existence of the ELABORATE pragma is already given in 6.12. Could these two requirements be joined? Since 1024 compilation units are allowed in a program, why the requirement for only 512 ELABORATE pragmas?

T030106 3.1.6 Ada is a free format language. There is no reason to impose a limit on the length of a line. A limit on the length of an identifier is reasonable, however.

- T030209** 3.2.9 Does this mean distinct literals? A limit of 1024 literals in a compilation unit prohibits all 4096 potential declarations from using an initial value.
- T030309** 3.3.9 3.3.9 and 3.3.10 cannot be tested separately. Any declaration introduces another identifier.
- T030405** 3.4.5 3.4.5 and 3.4.6 (as in 3.5.1; declaration vs use) are the same thing.
- T030706** 3.7.6 3.7.6 and 3.7.7 could be stated as one requirement since they are hard to separate.
- T030709** 3.7.9 STRING is by definition an ARRAY of CHARACTER. The maximum size of an array is already given in 3.7.4. Also, 3.7.8 already states that the maximum size of an object in bits is 65535. A string with 65535 elements would be at least 7 times as large as this.
- T030801** 3.8.1 In order to have 128 operators in an expression, you need at least that many primaries (3.8.3), i.e., 3.8.3 is redundant.
- T030802** 3.8.2 A function call is also a primary (3.8.3).
- T040103** 4.1.3 To facilitate testing, this requirement could be separated into two parts.
- T040104** 4.1.4 What does "implement an option to recover" imply? The degree of satisfaction of this requirement is subjective.
- T040202** 4.2.2 Is the position of the messages an option to the user or just the presence or absence of the messages?
- T040203** 4.2.3 4.1.6, 4.2.3, 4.2.4, and 5.1.2 are all similar. They could be combined or grouped together.
- T040207** 4.2.7 An Ada Source Statement has been defined in 1.1. So what are "a. Number of statements" and "b. Number of source lines"?
- T040302** 4.3.2 This requirement is very subjective. What may be sufficiently informative for the expert user may not be so for the novice.
- T040303** 4.3.3 A compiler may have a different set of error classes than given here. If so, it may not be clear if they perform the same function. In particular, some of the classes may overlap or split the ones listed here.
- T050102** 5.1.2 An absolute assembly code listing is not important when the target machine is a virtual operating system.
- T050203** 5.2.3 "Partial linking" needs to be defined clearly. This requirement is application and host/target specific.
- T050204** 5.2.4 This requirement is application and host/target specific.

- T060203** 6.2.3 6.2.1 and 6.2.3 are conflicting. Comments (6.2.3) are also Ada program text (6.2.1). If the reference to comments in 6.2.3 is omitted, this requirement is correct.
- T060307** 6.3.7 6.3.7 should be stated before 6.3.6, since 6.3.6 cannot be tested if 6.3.7 fails. You cannot specify the values of MIN_INT and MAX_INT in one representation clause without using non-contiguous integer code values, unless you define an enumeration type with an element corresponding to every integer from MIN_INT to MAX_INT.
- T060502** 6.5.2 It may be easier to test this requirement if it is restated to say "call and return" instead of "call or return", since an implementation may place all of the overhead for a subprogram invocation on one or the other.
- T060504** 6.5.4 Is this requirement intended to test the PRIORITY pragma or to test the fairness of dispatching tasks with equal priorities? These are separate requirements and if both should be tested this requirement should be separated into two requirements. The presence or absence of a pragma to specify time slicing may also be a factor.
- T060505** 6.5.5 Define "performance". What is meant by a task that is "not activated"? The declaration or dynamic creation of a task causes it to be activated. Before this happens the task doesn't exist.
- T060900** 6.9 To facilitate testing, this requirement could be separated into two parts.
- T061101** 6.11.1 Define "limit or restrict". Is this requirement the same as 6.3.1?
- T070500** 7.5 This requirement is fundamental to the production quality of a compiler. Unfortunately, it may be impossible to verify for compilers with inadequate internal development documentation.
- T080100** 8.1 A comment on the quality or content of the manuals and reports required by Section 8 may be in order.
- T080700** 8.7 The request for a maintenance manual indicates that the customer intends to perform their own maintenance. This is typically not the case for off-the-shelf compilers. A method for deciding when tests are applicable to a specific compiler should be formally outlined in the *Definition*.

F. Test Examples

This appendix contains examples of several features of the PQAC test suite and support software. In particular, an example of the code expander tool and a test example are given.

Each requirement assigned a test number in Appendix A is given a test file. This test file contains a statement of the requirement with a proposed method of testing that requirement. Included in the test file is any Ada test code required as well as special instructions for executing the test. More than one body of Ada code may be compiled and executed from each test file, allowing for comparison of compilation speed, execution speed, and code size. Compiler options such as OPTIMIZE_TIME may be specified for each compile.

Prior to execution, each test file is parsed to extract the test code and special commands. If the parsing program encounters `--* BEGIN compiler_name_1, compiler_name_2, etc.` then any text between this and the next `--* END` is simply flushed when the current compiler is not one of `compiler_name_1, compiler_name_2, etc.`

F.1 Code Expander Description

Some of the tests require large Ada files (e.g., testing that the number of allowable Ada source statements in a compilation unit be at least 4096). For this reason, a tool was developed to automate the generation of Ada programs containing repeated and similar constructs. The tool is written in Ada and recognizes a small set of symbols embedded in Ada text that indicate how to expand the code.

The operation of the code expander tool may be best explained using a simple example. Lines 1 - 14 of the given example show how the input to the tool appears. Lines 15 - 35 contain the output. The major functionality of the tool is its ability to encase code fragments within a nested loop structure allowing a very large test program to be stored compactly.

The symbol `--| EQUATE` is used to equate a name to a value. In this way the name can be used in the loop syntax. Simple addition, subtraction, multiplication, and division may be used. See lines 1 and 2.

Since Ada does not use '[' or ']' in its syntax, [X] is used to denote loop variable placeholders. In the above example, [1] denotes the outer loop variable, while [2] denotes the inner loop. See lines 6, 9, and 11. These counters may be offset by an integral amount inside the loops by using [X+1], [X+2], ..., [X-1], [X-2], etc. See line 9.

```

01: --| EQUATE iters IS 3
02: --| EQUATE size IS iters * 2 -- i.e. size is 6
03: begin -- some Ada block
04:
05:     --| LOOP iters [1]
06:     procedure proc_[1] is -- Note the name changing.
07:     begin
08:         --| LOOP 2 START 10 STEP size-1 [2]
09:         i := [1+2] + [2];
10:         --| END [2]
11:     end proc_[1];
12:
13:     --| END [1]
14: end; -- of block

```

..... becomes

```

15: begin -- some Ada block
16:
17:     procedure proc_1 is -- Note the name changing.
18:     begin
19:         i := 3 + 10;
20:         i := 3 + 15;
21:     end proc_1;
22:
23:     procedure proc_2 is -- Note the name changing.
24:     begin
25:         i := 4 + 10;
26:         i := 4 + 15;
27:     end proc_2;
28:
29:     procedure proc_3 is -- Note the name changing.
30:     begin
31:         i := 5 + 10;
32:         i := 5 + 15;
33:     end proc_3;
34:
35: end; -- of block

```

On each iteration of a loop, these variable placeholders are replaced by the current numeric value of an implicit loop counter. If an offset has been specified, then it is added to the implicit loop value. The symbols --| LOOP and --| END are used by the Code Expander to delineate the loops. A simple loop syntax is used to control the number of iterations (LOOP), the starting value of the loop counter (START), and the loop counter increment (STEP). See lines 5 and 8.

As we can see in the example, line 9 is nested inside two loops. The outer loop repeats three times and the inner loop repeats twice so this line gets modified and repeated 6 times in lines 19, 20, 25, 26, 31, and 32.

F.2 Test Description

This section contains an example test. If there is any Ada code contained after the beginning comments in a test, then this code is automatically compiled. It is not linked and executed unless the compilation name is specified in the appropriate place in the test file.

Text for test *T030305*:

```
-- T030305
--
-- depth of nesting of if statements = 256
--
-- Method:
--
-- Compile a procedure containing 256 nested IFs.
-- The compiler shall be determined to have passed this
-- requirement if the compilation and execution succeeds
-- without error.
--
--* EXECUTE Test_T030305
--| EQUATE iter IS 256

WITH Result;
PROCEDURE Test_T030305 IS
    Choice : Integer := 0;
BEGIN

    --| LOOP iter [1]
    IF Choice < [1] THEN
        --| END [1]

        Choice := 2;

    --| LOOP iter START iter STEP -1 [1]
    END IF;
    --| END [1]

    Result.Passed( "T030305", 100 );
END Test_T030305;
```

After parsing and executing this test, the output produced is:

Test Number T030305 (Compiler : DEC VAX, Host: VAX 8600)

depth of nesting of if statements = 256

Method:

Compile a procedure containing 256 nested IFs.
The compiler shall be determined to have passed this requirement if the compilation and execution succeeds without error.

```
----- TEST CODE -----  
--| EQUATE iter IS 256  
  
WITH Result;  
PROCEDURE Test_T030305 IS  
    Choice : Integer := 0;  
BEGIN  
  
    --| LOOP iter [1]  
    IF Choice < [1] THEN  
    --| END [1]  
  
        Choice := 2;  
  
    --| LOOP iter START iter STEP -1 [1]  
    END IF;  
    --| END [1]  
  
    Result.Passed( "T030305", 100 );  
END Test_T030305;  
----- END OF TEST CODE -----
```

```
Compiling ...  
Linking ...  
Executing ...
```

Test T030305 PASSED 100%

The success or failure of each test is automatically recorded. If a test requires manual intervention, such as examining a machine code listing, then a message to that effect is generated.

G. PQAC Test Suite - Volume II

A description of the PQAC test suite with a list of the tests developed is contained separately in Volume II. Procedures for rehosting and executing the test suite are also provided in that document.