AN ADA REVIEW:

A History , Problems and Ccmplaints ,

and the Current Status of the Language

by

MICHAEL IRWIN HODGES

B.A. , University of Pacific , 1975

M.S. , University of Southern California , 1978

A MASTER'S REPORT

Submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan , Kansas

1983

APPROVED BY:

PROFESSOR WILLIAM HANKLEY

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter I

HISTORY

1.1 INTRODUCTION

This repcrt is a review of the programming language called ADA. It covers three areas: the history, the complaints, and an overview of the current status of the language. ADA is making a significant impact on both governmental and private agencies. This is because Department of Defense (DOD) has mandated that all subordinate departments and projects under their auspices will consider the use of ADA where practical. This requirement has caused all who are a part of, and many who deal with DOD to initiate the use of a language that has not been fully tested or accepted by standardization organizations. Indeed the language has yet to see a complete working implementation.

Organization of the report will be as such. Chapter one, the History, covers the development of the ADA language within DOD. It requires no extensive language or computer

background. Chapter two, Language Complaints, is technically oriented and requires prior knowledge of language design and an in depth understanding of many computer science concepts. Some of the complaints are presented with a background and explanation of the feature and in some cases my personal comments about the complaints are given. Chapter three gives a representation and overview of the known status of the ADA language development. It also requires a minimal background for understanding. Provided as tabular data are all known U.S. and International ADA efforts as well as all known pertinent ADA publications.

1.2 THE NEED FOR LANGUAGE STANDARDS

In the early 1970's DOD was confronted by several significant problems in the computing area. DOD was annually spending about three billion dollars for software with projected costs expected to rise. Several intensive studies were completed that identified language proliferation as one of the major factors in this drastic increase(CARL 81). Many system development projects were using custom languages and compilers were being locally optimized for very specific applications. Many failures were resulting in these

development projects for a variety of reasons and interoperability and portability were virtually nonexistent. This sad state of affairs provided the impetus for DOD to institute the high-order-language-standardization program. Its goal was to standardize on a few well-specified, well-maintained languages to ensure the availability of high quality, production engineered compilers for computing within DOD.

1.3 THE HIGH ORDER LANGUAGE WORK GROUP (HOLWG)

The HOLWG was formed in 1975 to achieve standardization and end proliferation of languages within DOD. The first official position of the group was announced in the form of a DOD directive number 5000.29. This required that all subordinate agencies would utilize High Order Languages(HOLS), and that there would soon be an approved list of HOLS. Each would be assigned a control agency who would insure standardization within DOD. This directive was soon followed by DOD directive 5000.31 which was termed an interim list of DOD High Order Languages. The approved

languages were; Fortran, Tacpol, CMS-2, Jovial J3 and J73, COBOL, and SPL/1.

Concurrently several analyses disclosed the following points(GLAS79).

1. The first being that it was impossible to single out different sets of language requirements for different user communities.
2. It was also determined that it was possible to define one language to support all applications areas.
3. Further economic analysis (CARL 81) disclosed a potential software savings by DCD of one hundred million dollars per year by going to a single common language.

These ideas began to manifest to the point that the HOLWG initiated the defining of their perceived needed common language.

1.4 DEFINING THE COMMON LANGUAGE

Between 1975 and 1977 a series of requirements documents were produced. Each one had evolved from the critique and evaluation of its predecessor. These were

called; Strawman (USDD75), Woodenman (FISH76), Tinman (USDD76), and Ironman (USDD77). Their review and evolution to the next higher level was accomplished by circulating the documents throughout the United States and the European computing communities for comments and recommendations. When the design was sufficiently codified twenty three existing languages were evaluated to determine if any could satisfy the requirements for the common language. See Figure 1. However none of these existing languages satisfied more than 75 percent of the requirements and none were felt to be expandable into a language that could satisfy the common language requirements.

1. COBOL
2. FORTRAN
3. JOVIAL J73
4. JOVIAL J3
5. CMS-2
6. SPL/1
7. TACPOL
8. HAL/S
9. PL/1
10. ALGOL/68
11. CORAL/66
12. PASCAL
13. SIMULA 67
14. LIS
15. LTR
16. RTL/2
17. EUCLID
18. MESA
19. MODUAL
20. PEARL
21. MORAL
22. EL-1
23. PDL

Figure 1: CONSIDERED LANGUAGES FOR DOD COMMON LANGUAGE

In August of 1977 four contracts were awarded for competitive language designs. These went to Softech Inc., Intermetrics, SRI-International, and CII-Honeywell BUll. Softech and Intermetrics were well known in domestic language and compiler circles. SRI-International, which had evolved from Stanford Research Institute in California, was more of a surprise but explainable. The fourth company CII-Honeywell Bull was a French affiliate of the U.S. based Honeywell Corp. and was a complete surprise. The request for proposals, specified that the designs were to utilize one of three approved base languages; Pascal, Algol 68, or Pl/1. The resulting language designs were delivered in February 1978. All companies had chosen to use Pascal as a base language.

Pascal had become an extremely popular language for teaching software engineering methodology and structured programming. Its strength in these areas were in consonance with the requirements and the goals of the DOD common language.

The language design effort reached the pick-a-winner point in the spring of 1978. Eighty volunteer teams reviewed

the prototype designs. These teams were chosen on the basis of their interest from industry, government, and the academic community. Each of the four languages had been given a color coded name to insure anonymity of the contractor and was an attempt to minimize any prejudicial feelings. The four resulting languages were termed the Red, Green, Blue, and Yellow languages. Out of this intensive review the four contenders were narrowed to two. The two winners, which were felt to have a clearly superior design, were CII-Honeywell Bull, and Intermetrics. The languages chosen were the "Red" and "Green" languages. These two designs were consequently refined, and the language requirements were revised to become the Steelman document (USDD78). In May 1979, after four years development of a common tri-services and DOD wide language a final decision was reached. The government had selected "Green" as the winner, as defined by CII-Honeywell Bull.

Naming the language was not an easy task but in June 1979 it was finally decided to call the new language ADA. The name had been chosen tc honor the first programmer--Augusta Ada Byron the countess of Lovelace. She had been Babbage's first programmer. Babbage had developed the Difference Engine and the Analytical Machine more than 150 years earlier.

Between June 1979 and July 1980 the ADA language definition and Steelman document were converted into a "DOD Reference Manual For the ADA Programming Language", which was published in July 1980 (USDD80a). Between July 1980 and October 1980 approximately 80 typographical errors were corrected and other slight modifications implemented. The fruition of the DOD project was released in the form of the manual "Military Standard 1815, The ADA Programming Language" (USDD80b), in December 1980.

1.5 THE PROGRAMMING LANGUAGE ENVIORNMENT

During the ADA evolution it became apparent that a language is of little or no value unless there exists a programming environment in which to use it. Historically this has as a minimum meant compilers, link editors, loaders, debugging aids, and file system aids. DOD began asking the question, "Is that a sufficient definition of an appropriate enviornment?"

In the manner of the Strawman-Steelman, a prototype environment definition was called Sandman (USDD77a).

Subsequent to its first review a series of workshops were set up to review the environment definition and to focus on its requirements. In June 1978, under joint Army, Navy, and Air Force sponsorship, about sixty participants assembled and revised the original draft which was then called Pebbleman (USDD78b).

By August 1979 the next iteration of the definition of the programming enviornment was created in draft form. It was called Stoneman(BUST81). This was circulated and some revision clearly delineated the Ada Programming Support Environments. These were termed the ADA Programming Support Enviornment (APSE), the Minimum Ada Programming Support Environment (MAPSE), and the Kernal Ada Programming Support Enviornment (KAPSE). The availability of this modern, portable programming environment was expected to compliment the language in providing full software support for development of computer applications. The Stoneman requirements included encapsulation of the host operating system to improve portability of the APSES's; a central, unified data base for software objects; standard, well behaved interfaces for users as well as tools; an extensible tool set and command language; support for the entire software life cycle; support for large software

projects (not just individual programs); and strong support for configuration management (USDD80c).

Subsequently the Air Force let contracts to four companies for a "Competitive Design" of what they had termed the "Ada Integrated Enviornment" (AIE). My opinion gained from reading the literature suggests that they were taking control of the long range development of the APSE. The Air Force let a final production contract to Intermetrics to produce the AIE. The Army had been an ADA supporter from the early stages of development and felt that the Air Force AIE project would be slow in reaching fruition. It set out to implement an Ada Compiler and MAPSE into an existing Vax/Unix enviornment. In June 1980 the U.S. Army Communication Electronics Command (CECOM), awarded a contract to Softech to design, develop, document, and verify an Ada programming support enviornment called the Ada Language System (ALS), based on the Steeleman and Stoneman requirements (WOLF81).

Readily apparent is the parallel efforts by the Army and Air Force to create essentially similar products.

1.6 FROM HOLWG TO THE ADA JOINT PROGRAMMING OFFICE (AJPO)

The entire ADA effort for DOD and the HOLWG had originated under the direction of the DOD Management Steering Ccmmittee for Embedded Computer Resources (MSC-ECR). On December 10, 1979 the MSC-ECR commissioned a task force to develope a charter for a new orqanization to be called the "ADA Joint Program Office (AJPO)". This task force met during the spring of 1980 and established that the office would have a full time staff of six; a military Program Director, a civilian Technical Director, a secretary assigned by the Office of the Secretary of Defense (OSD), and three service Deputy Program Directors, one from each of the Military Services.

The package containing the charter for the AJPO was being considered by the Under Secretary of Defense for Research and Engineering (USDRE) at the same time the Military Standard 1815 was being approved. In no way was this a mere coincidence because Ada Byron had been born some 165 years earlier in the year 1815 and on December 10. The AJPO charter was approved and announced to the world on December 12 the same day that Ada Byrons birth announcement was published.

The purpose of forming the AJPO was logical. The formation of a new language that was sure to have such a large impact on the computing world, such that ADA was, needed a central point of contact, coordination, and guidance. It was these requirements that provided the impetus to establish the AJPO. With such a large and complex enterprise it was not a luxury but a necessity. This was the genesis for the language, its programming environment, and the office that now serves as the fulcrum for all development within DOD.

Chapter II

COMPLAINTS

2.1 INTRODUCTION

With an undertaking of such proportion and of so great a potential for diverse opinion questions concerning the language design and features were sure to surface. This was the first time that many modern language design features were being implemented into a large commercial language. Prior to the language becoming a Military Standard critique and recommendations were not so vehement. However, subsequent to it becoming a standard people in the computing world had the opportunity to see the completed manual, which defined the language and would be the genesis for the compiler development. Immediately voices of objection and discontent were surfaced.

Most of the complaints can be classified into certain themes. They have centered around the concern for the design of the language in terms of size, complexity, portability

and reliability, major complaints about certain features, and many miscellaneous points. These objections, problems, and recommendations will be subsequently presented in the following sections of this chapter by partitioning them functionly into the major divisions of; Lanquage Definition, Major Problems, and Miscellaneous.

The aforementioned ADA Integrated Enviornment, ADA Programming Support Enviornment, ADA Minimum Proqramming Support Envicrnment, and the Kernal ADA Support Enviornment will not be covered in this chapter.

2.2 LANGUAGE DEFINITION

BACKGROUND

The Military Standard describes the real-time programming language ADA, designed in accordance with the United States DOD requirements for use in embedded systems. Such applications typically involve real-time constraints, fail-safe execution, control of non-standard input-output devices and management of concurrent activities. Within DOD-1815 the stated design goals were; "a recognition of program reliability and maintenance, a concern for

programming as a human activity, and efficiency of the language". It further states that, "emphasis was placed on program readability over ease of writing." These two statements on the same page of the manual has caused some people to question what DOD really wants. However, when considering these statements they do not seem at all antithetical. If one considers the ease of writing code in APL and the difficulty in reading it you immediately see the DOD goal.

2.2.1 Size and Complexity

BACKGROUND

ADA is a large and complex language. In terms of shear numbers of keywords, operators, and functions it is fairly large. Pascal has 35 reserved words, FORTRAN 41, and ADA has 62. At first blush this does not seem to be a great obstacle but these figures are not a true representation. ADA keywords are defined as representing a particular syntactic construct and having a particular invariant spelling.

Additionally, there are syntactic constructs which conform to the above criteria, and which perform all of the duties of keywords, but are not included in the keyword list. (See Figure 2)

abort	digits	initiate	pragma	type
accept	do	is	private	
access			procedure	use
all	else	loop		
and	elseif		raise	when
array	end	mod	range	while
assert	entry		record	
at	exception	new	renames	xor
	exit	not	restricted	
begin		null	return	
body	for		reverse	
	function	of		
case		or	select	
constant	generic	others	separate	
	goto	out	subtype	
declare				
delay	if	package	task	
delta	in	packing	then	

Figure 2: KEYWORDS OF ADA

They include; END LOOP, IN REVERSE, IN OUT, RESTRICTED TYPE, IS PRIVATE, EXCEPTION RENAMES, END SELECT, SELECT WHEN, OR WHEN, IS SEPARATE, USE PACKING, USE RECORD, AT MOD, AND THEN, OR ELSE, and USE AT. There is no allowed variation for these as there is with a BEGIN IF or a BEGIN CASE statement. This really adds an additional 20 keywords for an actual total of 82. B.A. Wickmann (WICK82) in a comparison of ADA and Pascal has stated; "The ADA language is five or six times the size of Pascal. One can see this at a superficial level by counting syntactic productions, pages in the manual, or the number of lexical units. All indications are that compilers will be five or six times the size of Pascal compilers (given comparable code quality). At a deeper level one can enumerate the facilities that ADA contains that have no equivalent in Pascal."The follcwing figure gives a picture of the two languages and compares their size.

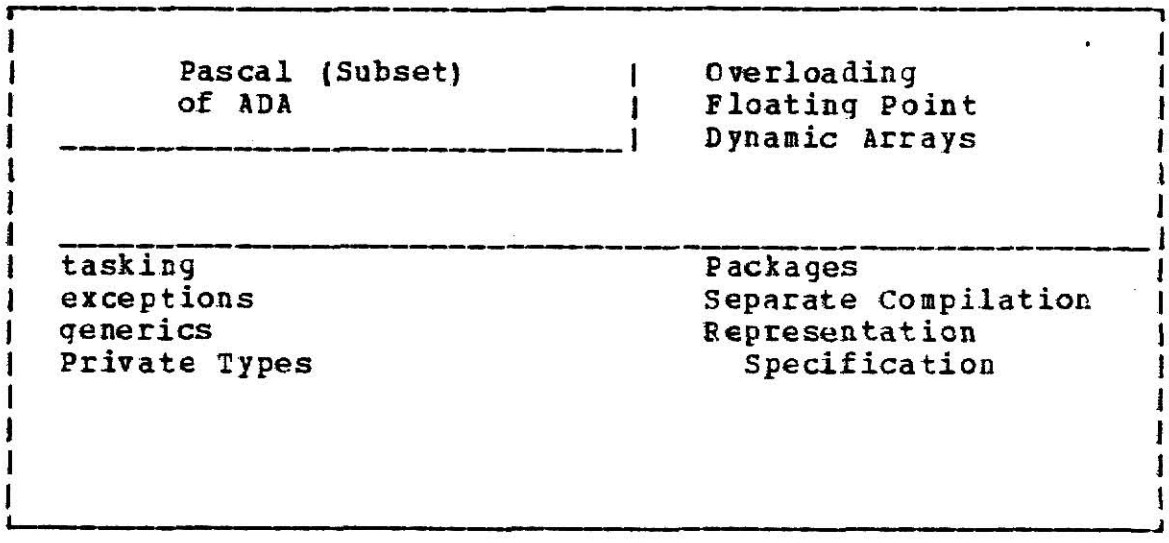

Figure 3: ADA vs PASCAL

Above the dotted line are features which 'could' be added to a Pascal language without radical revision. Below the line are facilities of ADA which have a major influence on the whole language design. This in itself is not overwhelming in terms of complexity but it does give credence to stated compiler implementation problems, learning complexity, and the potential for errors during compilations and computations.

COMPLAINTS

According to C.A.R. Hoare (HOAR73), "The language designer should be familiar with many alternative features designed by others, and should have excellent judgement in choosing the best and rejecting any that are mutually inconsistent. He must be capable of reconcilling minor inconsistencies or overlaps between separately designed features. He must have a clear idea of the scope and purpose and range of applications of his new language, and how far it should go in size and complexity...one thing he should not do is to include untried ideas of his own. His task is consolidation not innovation."

In light of Hoare's thoughts and the design goals of DOD-1815, it seems almost an impossible task to design a language that is sufficiently rich in constructs to handle universal applications, sufficiently complex to support concurrent and real-time applications of embedded computer systems, and that meets the purist interpretation of good design criteria. The problem of size and complexity of the ADA language has wrinkled several brows and fostered deep reservations about the reliability and cost effectiveness of the language.

Ledgard (LEDG82) has attacked the size and complexity issue and has campaigned publically for an official sub-set of the language. He has stated, "ADA is perhaps the most ambitious programming language project ever attempted. Nevertheless, despite its well intentioned goals, the size and complexity of the language continue to be the most significant technical obstacle to its success."

Ledgard has provided his own idea for scaling ADA down to look somewhat more like a concurrent Pascal look-alike. But DOD has flatly refused efforts for sub-setting ADA and has taken the position that any sub-set would lack the features necessariy to handle a comprehensive variety of applications and would defeat the purpose of going to a universal language. Ledgard and others continue to fight for a sub-set and have presented the following as potential benefits of a sub-set;

1. Cost of validation reduced
2. Tendency for sub-sub-setting reduced
3. Language forms are easier to remember
4. Development of conceptual model for using language easier
5. Likelihood of errors is reduced
6. Diagnostic messages are less confusing

7. Standardization efforts eased
8. Implementation less error prone
9. The language effort is less criticized
10. Users need not learn as much irrelevant information
11. The development of automated tools is simpler and cheaper

Ledgard has elucidated his sub-set ideas in (LEDG82) and depicts what he would delete from the language to obtain a smaller and less complex design.

COMMENTS

It is my opinion that with a language that must meet almost universal applications it must necessarily have many more features than a special language. Programmers need not learn or use all aspects of the language but may really only have a working knowledge of an applicable sub-set. I also take exception to Professor Hoare's comment about not including new and untried features in the design of a language. If we do not include new designs and features what is the worth of developing them and if not ever tried how will programming languages continue to evolve? I believe

that the ADA approach will greatly enhance the evolution of programming languages.

2.2.2 Uniformity , Portability , and Reliability

Bell Laboratories in (BENN82) an article entitled the "Hidden Costs of ADA" has taken exception to DOD's feeling that they are on the road to saving money with ADA. This saving is questioned in terms of uniformity, portability, and reliability in the total context of complexity.

2.2.2.1 Uniformity

BACKGROUND

Uniformity in the design and implementation of a programming language is important to the actual cost of using the language for a variety of reasons. Learning a language easily and quickly demands that programmers can have an intuitive grasp of both concept and detail. The ability to design, implement, or maintain code requires that the number of details one must keep in mind be minimal.

Uniformity also increases the efficiency and conciseness of the translator because the more contexts in which a construct can appear, the more often a piece of code can be re-used.

COMPLAINTS

In ADA negative exponentiation of left integer arguments is restricted. However in computer mathmatics both negative and real exponents are well defined. This lacks uniformity and does not increase the efficiency of the language and is in short a logical break in uniformity. The following 5 items are more examples of a lack of uniformity in the language definition (BENN82);

1. Underscores are permitted as part of identifiers. They must be isolated and must not terminate either end of the identifier. This makes scanning an identifier more complex. No ambiguity would result from lifting this restriction. An identifier like FIRE_Control is no more problematical than _FIRE_Control, and neither is worse than FIRE__Control. If this was done for readability it seems that they've missed the boat by not restricting multiple alphabetics, as in BACKVVVVVVVFIRE and BACKVVFIRE.

2. Underscores are permitted within numbers: they must be isolated, they may not begin or end a number, and they are not significant. This does not seem to add to readability but the costs of scanning, and uniformity of enforcement seem to add complexity to the language. i.e. 9387.7044 vs 9_383_7.7_044

COMMENT

The European community references numbers in the fashion 92_837.704_40 to preserve the place values and hopefully to reduce errors. This appears to be a good feature with some utility.

3. "a>b>c" may not be legal depending on whether the types of a, b, and c, are boolean or numeric. What is unexpected is that it is illegal for numerics and legal for booleans despite the fact that the almost universal purpose for these operators is to compare numeric magnitudes. This is because relationals return boolean and booleans are defined as to relative magnitude (false is "less" than true).

COMMENT

This if added would be contrary to every programming language commonly used today. The increased complexity and burden to the compiler do not seem worth the effort to reduce problems for naive programmers.

4. Floating point exponentiation restricts the exponent to an integer type without justification or mathmatical basis. This will cause local problems when programmers want cube roots.
5. ADA does not permit an expression composed of mixed logical operations even though they are totally unambiguous. Thus "a AND b OR c" is illegal while "a AND b AND c" is legal as is "a + b - c".

COMMENT

Other people have indicated that they are in fact ambiguous. What does the compiler do when given the statement AuB=>C? These are mixed logical operators and seem to confuse the issue considerably.

2.2.2.2 Portability

BACKGROUND

Portability is fundamental to the success of ADA. Portability actually encompasses: program portability (the ability to move an application from one machine to another), programmer portability (the ability of a programmer to move to a new site with minimal retraining), and compiler portability (the ability to move an ADA translator from one machine to another).

PROGRAMMER PORTABILITY

BACKGROUND

An ADA programmer can move much easier to a new site than a programmer using an assembly language or special military dialect. However, there are a suprisingly high number of ADA features which can vary from site to site even within the same machine architecture. These are some language parameters which are not constrained by the reference manual:

1. The maximum number of identifiers in an identifier list

2. The maximum length of an identifier
3. The number of digits allowed in numeric literals
4. The maximum length of a string constant
5. The maximum number of enumeration literals
6. Priority Range
7. All Pragmas utilized and authorized
8. The size and representation of numbers
9. Units of time
10. The maximum number of indices in an array

COMPLAINTS

1. Some of these parameters will likely be established by the compiler writer, and some by the installation manager. The distinction is not made by the ADA reference manual, and so will vary from compiler to compiler.

   COMMENT

   Other languages suffer from the same problem. This does not make ADA better or worse than other languages.

2. In their paper on the probable impact of ADA, Kling and Scacci(Klin79) have pointed out the extra costs incurred in training personnel when the language is very complex, and programmers have high mobility and low motivation. They question how productive software applications will be in view of the complexity of the language and the turnover and motivation of the DOD programmers.

PROGRAM PORTABILITY

BACKGROUND

There are a number of features of ADA which, if used, are capable of constraining a program to a specific installation or machine. Most of these appear in ADA because of Steelmans requirements for efficiency, both in run-time and in space. This requirement cannot be lightly put aside in the realm of real-time, process control software. If such a program does not meet or exceed its real-time constraints, then it is unacceptable. To meet these restrictions high utilization of machine resources is mandated. This implies a close bonding between machine functions and programmer

intentions which will almost undoubtably reduce portability. It has been demonstrated that certain ADA constructs will not provide similar results on different machines.

COMPLAINTS

The following examples will help to demonstrate this point (BENN82).

1. The use of tasks is guaranteed by the reference manual to have the same semantics on all implementations. However, there is no guarantee that all implementations will have the same efficiency. If intertask communications are unacceptably slow on a new target machine then redesign will have to take place.
2. Exceptions are not treated uniformly by all processor hardware. A system which relies on its safety on various overflow conditions may run erroneously on another architecture and give no indication of that fact.
3. Address specifications are not dangerous taken "in vitro". However, they may be used to derive information directly from the underlying hardware.

When this information becomes central to an algorithm, then the program will not move to another machine.

4. Machine code insertions are obviously machine dependent. If an algorithm is so time-critical that machine code insertions are required, then substantial changes are usually called for if the program is transported. The danger of this feature is that it will be used unnecessarily. Frequent insertions will make a program impossible to transport.

5. Interfacing to other languages presents similar problems to those of machine code insertion. The intent of this feature is to take advantage of capabilities outside of the milieu of ADA. These capabilities are often very site-specific. Since an interface to other languages is a very powerful capability, it will be an overwhelming temptation to couple ADA technology with other complimentary technologies. Unfortunately, the resultant hybird will not be exportable.

COMMENTS

My opinion of the complaints contained in this section is that they're all very true to varying degrees. But, the question that surfaces is; "Should we not allow these features in the language because they will erode portability or should we keep them, be cognizant of the problems and utilize them with constraint? The latter approach seems appropriate.

COMPILER PORTABILITY

BACKGROUND

The central issues in compiler portability are the complexity of the language and the requirement for highly efficient target code. ADA features such as multitasking, require extensive run-time support which will need much tedious, machine specific work to integrate into an enviornment.

COMPLAINTS

1. While the tedious, machine specific work, is probably less than that required for the design and implementation of an entirely new language, it will occasionally be greater. (BENN82)
2. Large projects will be able to justify retargetting an ADA compiler, but medium and small projects will not. The costs will be prohibitive (BENN82).
3. The current practice of using whatever translator can be made available cheaply which satisfy project needs, will likely be more attractive economically than the retargetting of an ADA compiler.

COMMENT

These three complaints seem to be trivial. Other languages suffer from the same problems. The complaint might be that ADA has not solved this problem.

2.2.2.3 Reliability

BACKGROUND

The production of reliable code is important to the development and maitenance of software. Zelkowitz (ZELK78) estimates that for large systems the cost of software production and maintenance increases four times faster than the complexity of the design. If reliable code is difficult to produce, then many of the errors will be obscure and will not show up until the testing stage of the system development where the cost of fixing them will be five times greater than it would have been in the coding stage (ZELK78).

COMPLAINTS

1. According to Bennett (BENN82) there are a number of factors that enter into the production of reliable code. One of the most important is the complexity of the language. He states; "Programmers tend to make more errors as the language becomes more complex. Futhermore, the reliability of a compiler decreases (almost geometrically) as complexity increases, due to the vast amount of complicated syntax and semantics it must translate. As a language becomes

less uniform, the programmer makes more errors because he no lcnger can rely on his intuition about the syntax and semantics of data structures, logical structures, and the like."

COMMENT

At first blush his comment about programmers making errors seems logical. But in ADA and other structured languages features are provided to help reduce errors. A typical example is strong type checking which greatly reduces errors even though the declaration of types is a powerful and sometimes complex issue. So I think in some cases the language introduces complexity and power but has checks and safety features that actually reduce run time errors. His comment seems naive. I also question his comment about the reliability of a compiler decreasing geometrically as complexity increases. I have no background in this area but question even the logic of his statement. Example; Is FORTRAN more complex than BASIC?. If so, ergo, FORTRAN compilers (using his logic) must necessarily be less reliable. I think not.

2. Two studies in software reliability performed at Boeing (GLASS79A, GLASS 79B) revealed aspects of verifying and debugging code. Conditional compiliation was second only to peer code review as an important criterion for the production of reliable software. Yet ADA does not possess this capability, even though it is the cheapest of any capability to implement. The ADA manual states a good ADA compiler will not compile a piece of code which is not executable. This according to Bennett (BENN82) gives the programmer the illusion that ADA has conditional compilation capabilities. He states, "Optimization of this sort is highly context sensitive. Optimization is not required by ADA and may not be implemented. This type of "conditional" compilation requires extra editing, inserting, and deleting of the source code, by the programmer, which increases the likelihood for errors." (Inserting debug statements in the wrong place, deleting the wrong lines, etc.)

COMMENT

Conditional Compilation is referenced by Bennett (BENN82) and Brosgol (BROS82), neither have provided an explanation of their definition. ADA provides separate compilation of program units and also has the ability to compile compilation units with the declaration section only without the body of the unit being written. The program library retains all information of compiled units and only must recompile those units that are affected by a newly compiled unit. So to build large programs only the declarations need be written at the outset to check program flow. Then the program bodies (stubs) can be filled in. I am not sure what the root of this complaint is. If he is indicating that a sub-set compiler can be used as needed he is correct in that ADA does not provide this facility.

3. Even though I stated I would not discuss environment this complaint is considered as referencing the enviornment and not the language. A means of producing reliable code, according to the GLASS studies, and reinforced by Ghezzi and Jazayere (GHEZ82) is by using other source code debugging

techniques such as data tracing (tracing the changing values of a variable), logic tracing (tracing logical paths of execution through the program), and assertion checking (checking that specific conditions are met). ADA does not provide the capability for data tracing or assertion checking, and does not give the programmer the capability to reference previous values of variables. Problems occur if the compiler writer wants to include these features, he in essence is creating a superset of ADA, which is expressly forbidden (BENN82).

COMMENT

These facilities are planned for the APSE and are not planned or desired to be included in the language.

2.3 MAJOR COMPLAINTS

Perhaps the most critical admonition to the ADA language came during a speech, in 1980, by C.A.R. Hoare (HOAR81). He was speaking at the Association for Computing Machinery's presentation of its highest award for technical

contribution. He cited many previous problems he had encountered and lessons he had learned from his years of experience. He stated;"None of the evidence we have so far can inspire confidence that ADA has avoided any of the problems that have affected other complex language projects of the past. The original objectives of the language included reliability, readability of programs, formality of language definition, and even simplicity. Gradually these objectives have been sacraficed in favor of power, supposedly achieved by a plethora of features and notational conventions, many of them unecessary and some of them, like exception handling, dangerous." In this section the most vehement complaints from the computer community will be examined. These however are specific semantic issues and are not complaints because they add to the language size.

2.3.1 Exceptions

BACKGROUND

Exceptions are one of the controversial items of the language. Exceptions provide a mechanism for the unusual termination of program units. Normal termination of blocks

and procedures occurs by executing the last statement of the body, while normal termination of functions is by executing a return statement. Exceptions provide a dynamic mechanism for exit from program units which bypasses the above normal termination mechanisms. Exceptions may be defined by the user or predefined by the system. Predefined exceptions may be illustrated by INDEX_ERROR (when an index is outside the range specified by an array). Since INDEX_ERROR is a predefined exception there is a sytem-defined "default" action when an index error occurs. However, this system-defined action may be superceded by user-defined actions to handle index errors in specific parts of the program. Consider the following example.

When an exception is raised during execution of a subprogram, a handler is sought first local to the subprogram, and then in successive dynamically preceeding program units. This dynamic criterion for determining the handler associated with a raised exception contrasts with the static scope criterion of associating a procedure with a procedure call. An exception is regarded as an error. The enviornments in which the exception occurs and the environment in which it is handled are always abandoned.

```
                                --Two dashes mean comment

Procedure P is
 Singular: exception                --Declaration of Exception
   Procedure Q is
     begin ...
        if determinant = 0 then --code in Q which raises
           raise Singular;      --the exception Singular
        end if;....
   end Q;


   Procedure R is
     begin...                                --if call of Q within R
       ..Q..                                 --exception occurs code
       exception                             --of 2nd handler used
          when Singular=>--#2 Exception Handler
   end R;


begin--P
   ...R..Q...            --if call of Q during main body
      exception          --of P exception handler #1
      when Singular =>   --#1 Exception Handler

  end P;
```

Figure 4: EXAMPLE OF AN EXCEPTION

COMPLAINTS

The exception handler is a mechanism for unusual completion of the subprogram in which the exception is handled. The following is a list of points against exceptions and their use.

1. The detection of exceptions may be suppressed. For example, if a time-critical computation wishes to

save the time of checking for index errors in a given program unit, this can be accomplished by including the following pragma in the declarative part of the program unit. Pragma suppress (INDEX_ERROR). This allows the compiler to omit run-time checks for index errors in compiling the program unit but does not require it to do so. These are forms of optimization the compiler may use or ignore. Use of this facility can lead to uncontrolled program behavior when the suppressed exception occurs and is not properly handled, and because programs with suppressed exceptions may behave differently for different compilers (WEGN80).

COMMENT

The management control of the use of the suppress pragma will have to be very tight. This like many other features is potentially dangerous and will have to be managed and monitored very closely. Its use however seems to have utility.

2. Steelman requires that raising an exception transfers control to the most local enclosing exception

handler. "Enclosing" in this case can only refer to dynamic calling enviornments since unhandled exceptions in rcutines are re-raised "at the point of call in their callers." Thus, exception handling is tied to the dynamic chain , a fact that makes it inconsistent with the static scoping mandated for the rest of the language by Steelman and also inherently more difficult to verify than if it were done lexically (YOUN80).

COMMENT

With only a modicum of understanding I may not fully understand Mr Youngs complaint. However, it seems that whether the scope checking is inconsistent or not is irrelevant. What is important is that we do transfer control from a program body that has raised an exception to its appropriate handler. Also execution when the unit is terminated must necessarily pass to the calling unit-dynamically and not statically. I agree that this is more difficult to check and verify but it seems to be the nature of the beast.

3. A translator may choose to evaluate the constituent terms of an expression in any order that is consistent with the precedence properties of the operators, and with the parentheses. As a consequence, the order in which exceptions might occur in the evolution of an expression is not guaranteed by the language. The formal semantics of the language only defines the value of an expression whose evaluation does not raise any exception (RATD79).

COMMENT

This is a valid point but it seems that this is allowed for compiler optimization and restricting order would reduce efficiency and optimization.

4. The ability for one task to raise or to propagate an exception in another task must be viewed as a possibility with potentially severe consequences in parallel processing. In no way should such external exceptions be considered as being normal terminating conditions. Interfering asynchronously with the execution of a task may catch it in a state where it is not prepared to respond to such intervention. There is then always a risk of leaving the task in a state of confusion and also, of contaminating other tasks that were communicating with it. This can cause an inability to rendezvous, or terminate an accept when; an exception is raised within the accept statement, a third task disrupts the called task by an abort or failure exception, or a third task disrupts the calling task. All which are extremely complex and difficult to program handlers for due to the variety of error conditions (RATD79).

5. Professor Hoare, as previously cited, has expressed his belief that exceptions are extremely dangerous but has not chosen to share with us why. His reputation alone gives credence to his thoughts and only add to the plethora of ccmplaints about this facility(HOAR81).

The fact of not being sure of when exceptions will be raised based upon the requirements of real-time processing, especially in embedded systems such as missles, guidance control systems, and Command and Ccntrol have led many to question the prudence of using such a facility. When this facility is linked to a program package that has hidden elements, and encapsulated in a dynamic concurrent process, exceptions could arise in a multiplicity of ways. It is this uncertainty of outcome that has caused reservation about this facility.

2.3.2 Tasks and Task Communication

BACKGROUND

Tasks are the facility that provides parallel processing in ADA. Tasks are a textually distinct program

unit which may be executed concurrently with other tasks. A task is comprised of a specification part, which describes its external appearance, and the module body, which describes its internal behavior. Tasks must be initiated. They raise the Initiate_Error if raised while already active.

Entry specifications are the principal mechanism for communication and synchronization among tasks. Entry specifications are syntactically like procedure specifications. They may have parameters with the same binding modes as procedure parameters, and may be called from other tasks by entry calls which are indistinguishable from procedure calls. However, whereas procedure calls serve to immediately invoke the called procedure, entry calls require synchronization with an accept statement in the task body before they can be executed.

2.3.2.1 Concurrent Process Discussion

ADA provides communication between processes, called tasks in ADA, on the call level. That is, a process may

communicate with ancther process by calling a special type of procedure, the accept procedure (of which there may be several), which is part of the called process. For example, in a process1 we may have a call to process2.accproc(parms). The caller must use this two-level identification, but the accept procedure accepts calls from anybody. This is sometimes called many-to-one mapping.

Processes in ADA are always started and sometimes stopped explicitly by the initiate and abort primitives. Hence there are no implicit starts as a side-effect of the call or accept primitive. During the process call, parameters are passed to the called accept procedure. The calling process is suspended until the accept procedure (usually a compound statement) has been completely executed by the called process, and possible output parameters have been returned to the caller. This get-together is called a rendezvous. During the rendezvous the processes participating are thus implicitly synchronized.

A process may allow a certain sequence of calls by simply sequencing a group of accept procedures (they may carry the same entry point name). This is because accept procedures are not executed asynchronously, but they are executed when encountered inside the body of statements in a process.

In order to allow a choice of several accept procedures a variant of the case statement has been introduced, called the select statement. Every branch in a select contains an accept procedure. However, there are two exceptions to this rule. The first is that a delay statement may be used instead, the second is the else part of the select. Case branches in the select may furthermore contain when guards, boolean expressions usually containing local variables (but not always) which have to be true for the branch to be accessible. The else primitive indicates those accessible, because all guards are false, or if an immediate rendezvous cannot be made. If a select without an else occurs and all guards are false an error condition (select_error) is raised. Because select occurs in the middle of executable code, all ADA's control structures may be used inside and outside of it. For instance a select compound may be surrounded by a loop.

The basic difference between the select and the case statement is that if more than one branch can be taken (guards true and calls outstanding) a random choice is made for select, while for the case the branches must be unique and only one can be selected. This facility is much more complicated than presented herein but it is not feasible to

provide a complete discussion of this complex facility in this review. The following is a list of the major complaints about tasks and task communication.

COMPLAINTS

1. The ADA Rationale calls the rendezvous concept a notion at a higher order of abstraction than send's and receive's. This may be so but frequently a heavy penalty has to be paid for this. During a call both processes are suspended, except for the accept code which is executed during the call. Call parameters could be accessible via common storage, but in the case of a distributed computer network this won't do. If processes are executing on separate computers a better solution appears to be to handle parameter passing by means of messages through normal network I/O channels. In that case a call starts with a hidden send (of parameters) to the callee and ends with a receive in order to pick up updated parameters. During all the time the called process is executing the accept procedure, the calling process hangs, while in actual fact it could resume processing as soon as the parameters have been

accepted. In an environment of heavily communicating processes, this tends to make the system of processes behave more sequentially than necessary. In order to avoid this a call could be split in two sub-calls, one to pass the parameters to the called process, the other to return (output) parameters to the calling process. But this solution would implicitly lead to the somewhat lower level of sending and receiving messages, and would therefore run counter to the rendezvous philosophy (BOS-80).

2. Another criticism is concerned with the asymmetry of the ADA naming. ADA's choice appears to cover a large class of communication problems but certainly not all of them, at least not in a safe way. Why shouldn't the called process have the capability to single out its customers? The argument is that a user has to know its server, but that the servicing process need not know its customer. In many cases this argument is fallacious. Two examples follow.

   The first is ping pong communication, such as in co-routines. This could be done by having the one process repeatedly issue calls to the other process (the parameter list would include both input and

output parameters). This scheme would work, though at the cost of an almost complete serialization of the two processes. The situation is improved in an approach in which we split up the ping pong call in a call coming from the original calling process, and a call returned by the originally called process. In this situation reciprocal naming is in order, otherwise other processes might sneak in and activate the (non-discriminating) accept procedures.

The second example poses an even more serious problem. It goes as follows. Imagine a system of processes all routing their output to a single (physical) printer. We avoid interleaved output by dedicating the printer process to a customer until this customer is done (as indicated by an EOF). In ADA the only way to solve this problem is to have the user serialize the printer by embracketing the calls to the printer process by something like the P and V semaphore operations. A poor solution for a language which claims implicit synchronization. Even worse is that the situation is totally unsafe, because it depends on the user obeying a certain protocol, where in fact the protection should be enforced by the

service routine. This is how a strictly safe solution goes. The printing process, in recognizing the first line of output, determines the sender. Subsequently it goes into an accept loop only taking records from the sender, until an EOF arrives. At that moment it locps back to a situation where another first line may be accepted. It is clear that this solution is not possible in ADA(BOS-80).These same complaints are voiced by Silberschatz in (SILB81).

3. Another issue in ADA which raises great concern about simplicity and efficiency of implementation is the exception handling mechanism defined for tasks. This was touched upon lightly during the exception section. To present a clear criticism of this feature, the following example which represents a program segment of a task T1 which raises a failure exception in task T2 and aborts it after 20 seconds if by then it has not terminated itself.

```
If T2 ACTIVE then
   raise T2.FAILURE;
   delay 20.0*seconds;
   ABORT T2; --if it has not terminated yet
end if;
```

Figure 5: EXAMPLE OF TASK WITH EXCEPTION

The effect of the statement raise T2.Failure is to interrupt T2 if it is executing and force it to take appropriate actions. This is a deviation from the ADA approach which treats tasks as independent units communicating on mutually agreeable terms. While it is plausible that forceful means are sometimes necessary to prevent chaos and limit the extent of damage that can be caused by a misbehaving task, I feel that other language features are not totally compatible with this. For instance, in order for a task T1 to justifiabley raise a failure exception in another task T2 and eventually cause its termination, it must first have some means of detecting the misbehavior of T2. However, if T2's misbehavior is, for instance, its lack of accepting messages from T1,

then T1 can never detect the failure as it will be waiting indefinitely to rendezvous with T2.

2.3.3 Pragmas

BACKGROUND

Not as much has been opposed concerning pragmas as has been said about the other language features described in this section. However, almost all features are linked with the other problems thru the use of pragmas and together they compound or generate the problems. A short discussion of this feature is included.

A pragma (from the Greek word meaning action) is used to direct the translation system in particular ways. They are used to convey information to the compiler. They are like comments in that they generally have no effect on the computation performed by the program (but not always). But a pragma can have considerable effect on the information supplied to the user as a result of compiling or executing a program.

```
pragma LIST(on);  --Provides the user with
                  --a listing of the program
```

Some pragmas, such as that above, specify a definite obligatory action on the part of the compiler. Other pragmas, such as the ones in the next example, are suggestions to the compiler which may or may not be implemented.

```
pragma INCLUDE("name");  --substitute the text file named
                         --in the pragma at the program
                         --point where this pragma occurs

pragma SUPPRESS(exception names);
                         --run-time checks for the named
                         --exception may (but need not) be
                         --suppressed in the program unit
                         --where this pragma occurs
```

Figure 6: PRAGMA EXAMPLE

It is this above type action that is the point of contention. Some pragmas are defined by the language. See (USDD80b), Appendix B. It is expected that other pragmas will be defined as part of the support environment developed around the language. Again let us take a look at a couple of the specific complaints.

COMPLAINTS

1. Wegner (WEGN80) in his complaint of exceptions directly links his example of the exception problem with the suppress pragma. Pragma suppress (INDEX_ERROR) could cause the compiler to optimize and not check for any out of range or INDEX_ERRORS. This allows the compiler to save time by not checking but as several of these pragmas do, it does not require the compiler to do so. Different compilers could treat this feature in different ways. Different runs on the same compiler may also treat this feature differently if the choice for optimization is tied to dynamic operating system statistics. The bottom line is that program correctness and consistency of outcome is questionable utilizing the suppress pragma.
2. The include pragma may alter program behavior and outcome if embedded in a nested task that may be active before or after a required prerequisite task. Defensive programming and sequential methodology is the key to preventing the include pragma from going afoul. However, when you mix the task communications problems with the include pragma potential catastrophic results are possible. This feature is

also a form of conditional compilation which is counter to the design goals of the language (BROS82).

2.4 MISCELLANEOUS PROBLEMS

2.4.1 Association for Computing Machinery (ACM) Position

In April of 1981 a canvass was submitted by the standards committee to obtain a position on whether to recommend American National Standards Institute (ANSI) standardization for ADA. If ADA was found to meet various ANSI criteria the language would then be adopted as a dual standard probably designated as ANSI/DOD MIL-STD-1815. Requests by the ACM standards committee for comment from the members of ACM appeared in the May issue of SIGPLAN Notices with voting to conclude by October 1981. One hundred and fifteen members responded prior to the cutoff date: 39 in favor of ANSI standardization, 72 opposed, and 4 abstaining. This was the largest response ever to a question of standardization by the members of the ACM.

In the overall canvass process, ACM was one of 96 organizations responding to the canvass; 66 concurred with the proposed ANSI adoption, 23 objected, and 7 declared themselves "not voting." The analysis by AJPO of ballots and public review of comments resulted in a partitioning of 380 specific comments. The specific results of the complaints of the ACM against an ANSI standardization were published interleaved with the AJPO's answers in the February 1982, Communications of the ACM. Three major points were identified as the basis of the negative ACM vote.

1. Insufficient precision and detail of specifications
2. Lack of assurance of implementability
3. Failure to identify reliable and efficient subset(s)

ANSI standardization did not occur. AJPO as a result of the problems and critique planned chapter reviews in hopes of obtaining a more definitive specification and improved precision. This was to be a project to reach fruition at the same time that a compiler could be implemented and validated. This being in the context of achieving ANSI standardization for the ADA language by eliminating the majority of the complaints and proving ADA could be implemented.

Chapter III

CURRENT ADA STATUS

3.1 INTRODUCTION

In retrospect, Chapter one covered the years 1975 thru 1979, and provided the rational for the impetus and genesis of the ADA language. Chapter two, covered the complaints and concerns of the computing community subseguent to the release of DOD-1815. This in essence was from 1979 thru 1981. During this later period, and understandably so, a kind of acceptance of ADA was slowly taking place. It did appear to many that DOD had been quite stoic and unmoveable in their position of not subsetting and not modifying the language. However, to some degree, this has given a perceived stability in the emerging language and has to a point nurtured the development of the language implementations. As seen later in this chapter some change and modification was necessary and will be presented. An excellent example of this transitiCn of feelings was provided in an introduction to the article "A Methodology for Modular Use of ADA", (BEN-82). It stated, "At the

outset, we wish to make it clear that we regard the language as frozen and thus we will not add another article to the flood describing what ADA should have been or should become. If even the recipient of the Turing Award, Professor Hoare, admits his advice on simplicity, and subsetting went unheeded, we regard it as futile to continue with such criticism." The growing plethora of literature about ADA is beginning to become overwhelming. It comes as no surprise that many language and compiler problems are being encountered. The literature however, is turning away from criticism and is now zeroing in on solutions to problems, methods for circumventing deficiencies, and an overall acceptance that ADA is ADA. This chapter will cover the current ADA status as it is developing within; Department of Defense, Civilian Agencies, and International circles.

3.2 DEPARTMENT OF DEFENSE

3.2.1 Changes

Under Department of the Army contract number 4352-1, changes tc the ADA language were evaluated, recommended and

implemented by Intermetrics Inc. Although as of this date the revised manual has not been released Benjamin Brosgol (BROS82), has provided an explanation of the philosophy of the changes and examples of the scope of each change. Many of the changes and recommendations were a direct result of the many problems voiced by users, and implementers as expressed in the preceeding chapter herein.

The chief goal of the ADA revision process was to make only those changes that were necessary, and to do so in a very conservative manner. Four aspects of the language were changed: precision of definition, functionality, implementation, and teaching. Removeal of some complex and error-prone items eliminate some very difficult run-time issues. Listed below are a few highlights of the many changes.

1. The body of a generic subunit is in general not available at the instantiations. To simplify the implementation, it is now permitted that a compiler require the body for a generic subunit (as well as the bodies for its subunits, etc) to be present in the same compilation as the stub declaration. The same restriction is allowed for generic library units.

2. Name resolution in generic templates is now clarified to emphasize that name binding occurs prior to and independent of instantiation. (*stronger type checking- will not allow joining of different declared types-even though they return the same type*).
3. Recursive instantiation (Generics) is now forbidden.
4. Tasks: The Failure exception is now removed from the language, because it proved to be extremely error-prone and difficult to implement.
5. The 'terminated' attribute is not sufficient for the purpose of determining whether a task may accept an entry call, since it is possible for the task to be "uncallable" before it is terminated; viz., when it is awaiting the termination of its dependents. The 'completed' attribute is now included to reflect this case. Calling an entry of a task for which 'completed' is true raises tasking_error in the caller.
6. The Include pragma has been removed as a required pragma, for two reason. First, it was inconsistent to define Include in the language without giving similar status to source inclusion facilities for conditional

compilation and it was not desireable to add the latter complexity to the language. Second, Include can be provided by an implementation.

Most of the language changes will not effect the teaching of ADA, since the level of refinement is below the threshold of most ADA books or training plans. For a complete listing or reading of the changes see (BROS82). In summary, the language revisions are large in number but small in nature. The emphasis on language stability has served to reduce the effects of these changes. The AJPO has announced early 1983 as a publishing date for the revised manual.

3.2.2 Army

The Army has supported ADA almost from its inception. It has several projects of some size in various stages of completion and is making a growing number of commitments to ADA. At the outset of the ADA development both the Air Force and Navy took the 'wait in the wings' position in regards to ADA to see how the language would develop without wasting

their own money or effort. It is my opinion from reading available literature that the Army has greatly fostered the development of the language and its enviornment by spending large sums of money, time, and effort. In all fairness to the other Services they had spent quite a significant effort in developing their embedded languages and were satisfied with their performance.

The Army has for several years had an ongoing contract with Softech Inc. This contract encompasses the production of the full ADA compiler, a translator for ADA to Pascal, plus a MAPSE, all to operate on VAX and PDP/UNIX. In conversation with Mr. Pete Fonash of the AJPO, it was discovered that the compiler and the enviornment are virtually complete and are undergoing acceptance testing by the Army and also Compiler Validation Testing. Both of these projects implement the original ADA.

The testing suite for ADA was developed independently by Intermetrics Inc. and is being administered by the Mitre Corp. Mitre has responsibility for the ADA Validation Organization (AVO). It is AVO's responsibility to validate all ADA compilers and to publically report approved compilers and problems encountered when compilers fail validation. It has been projected by AJPO that the Army's

compiler and enviornment will be the first ever to gain validation. It has been projected that this will be accomplished early in 1983.

The Army also has another very large project in an unknown stage of completion. It is the "Military Computer Family". It suffices to say the project is is an attempt to standardize hardware configurations so that interoperability, interchangeability, and ease of replacement are enhanced. The plan is to have a standard computer used throughout the military. ADA has been chosen as the target language for these computers. The project is contracted by the Army to Litton Data Systems. The final completion date for this project has not been made public and it to date has taken on an evolutionary and continuing flavor.

New York University (NYU) is also under Army Contract to produce and executable semantic model for a full ADA. It is written in SETL and includes a compiler and interpreter that, respectively, generate and execute a tree-structured intermediate form. The initial compiler (1981) and system, which was @ 80% of the full ADA, generated code at @ 25 lines per minute. Since then significant refinement has taken place. Currently a subset ADA is available for the

VAX/VMS from NTIS, a VAX/Berkeley Unix version from NYU, and an AMDAHL UTS version from NYU. What is surprising is that all problems and complaints of the systems are supposedly to be referred to U.S. Army CECOM, rather than to NYU. Full ADA is projected to be complete in June 1983.

As part of the Army's wide ranging ADA program the investigation of systems design techniques utilizing ADA have not gone unnoticed. The U.S. Army CECOM and the Center for Tactical Computer Systems(CENTACS) at Fort Monmouth, N.J. awarded three contracts in 1981 to develop design techniques utilizing ADA. CDC was awarded a contract to redesign the existing AN/TSQ 73 "Missle Minder" system, General Dynamics was given the job of redesigning the An/TYC 39 message switch and Softech was contracted to oversee both of the approaches and develop educational materials from the results. These contracts were to be completed in late summer of 1982. To date no findings have been publically announced.

The Army has made a full commitment to ADA. It is using ADA for new projects as quickly as it can in view of the lack of a validated translator. The Army has over 96 embedded computer systems with many different sets of applications. With their large array of requirements and the

ongoing need to update and modify programs their support of ADA will surely help aid its proliferation.

3.2.3 Air Force

The Air Force has invested a significant amount of time and money into the research and devlopment of the ADA language and its enviornment. The most significant project heretofore is the ADA Integrated Enviornment (AIE), which is a Stoneman-conforming MAPSE. The system is divided into the MAPSE tool set and the KAPSE. The tool set contains the compiler, linker, editor, debugger, and a command language interpreter that processes an ADA-like yet user-friendly language. A Virtual Memory Manager is provided to give tools standardized access to structured disk-resident data. The KAPSE includes the loader, database manager, and ADA run-time system, as well as the host interfaces (for IBM VM/370 and Interdata OS/32). Intermetrics is the contractor for this project and the required delivery date is December 1984.

The Air Force has two other ongoing ADA projects, which include, a contract to Syscon Corporation for the

verification and validation of the AIE, and a contract with Software Engineering Associates for a Jovial/ADA Microprocessor study. The latter to be complete in August of 1982, the former June of 1984.

3.2.4 Navy

Within DOD the Navy has done perhaps the least of any of the Services in adoption of ADA or in research efforts to develop ADA. They, in short, had waited to see if the language was going to succeed. They have started to implement ADA but are proceeding slowly. The Naval Ocean Systems Center is heading the reasearch for the KAPSE Interface Team (KIT). The Navy is not developing its own APSE, but is leading the effort for culmination of standards and conventions for APSE interfaces. These will include the standardization policies for the ALS, AIE, MAPSE, KAPSE etc. The Navy is also procuring approximately three tools to be designed to run on both the Army's AIS and the Air Force's AIE.

The only known other Navy effort with ADA is a restructured Naval Tactical Data System (RNTDS). As part of the "now" generated overall Navy transition-to-ADA, the Navy

is studying the feasibility of changing an existing large shipboard command and control system to ADA. The system now suffers from being implemented on a number of different equipment configurations, from little standardization between configurations, and from the resulting increase in development and maintenance costs. Restructuring using ADA is expected to amelicrate this situation.

3.2.5 AJPO / Standardization

The office that supervises and directs the ADA efforts within DOD has really matured into a critical focal point for the ADA development. The AJPO permeates the literature regarding ADA and all coordination for DOD is channeled thru the office. As a general observation, from various readings, it appears the office is providing adequate direction and is meeting the purposes for which it was formed.

The AJPO has worked closely with the ACM on outstanding issues preventing ANSI standardization. As previously stated the manual, MIl STD-1815, has been revised. The first compiler and APSE are in the AVO

validation process and should receive validation in early 1983. A recent canvass of the ACM, disclosed in the August ADA-TEC, indicated that only 4 negative votes were preventing a unanimous approval of ANSI standardization. Two of the problems were in the design area and two were in the prose of the manual. All are being resolved and should be complete prior to the next formal voting. With the completion of the validation of the compiler ANSI standardization should and is projected to be forthcoming. This is projected to occur in early summer of 1983.

The AJPO, like the Services, has many contracts let for ADA projects. These include; A full ADA debugger and test suite (Intermetrics), an ADA Validation Oraqanization (MITRE), and several smaller projects. For a list of all known major ADA efforts in the United Staes please consult data at TAB 1. Other efforts to expand ADA include the inclusion of ADA as an approved DOD HCL. Also, all embedded systems projects must now consider ADA and if not used justify the non-use. It is my opinion that this is just the transition and prelude to mandatory use of ADA when production compilers and enviornments are validated and released. The AJPO has placed ADA notes and all current information pertaining to ADA on ARPANET (Host number 23dec)

or via Telenet (address 21389). The directory name is ADA-Information and the password is ADA. Another directory is ADA-Answer. The password is also ADA. It is designed to answer all questions concerning ADA standardization.

3.3 CIVILIAN ADA EFFORTS

There are many projects ongoing in the civilian communities involving ADA. Much of this has of course been spurred on by government contracts and fallout from the industry perception of the potential influence of the language. The ACM has formed a Special Interest Group on the language and publishes a bi-montly publication entitled, SIG ADA-TEC. This has been an excellent source of language information and greatly increases the understanding and status of the language. A few points of interest will be presented to show a representation of the progress of ADA in the civilian enviornment.

1. Many software producers have recently marketed ADA compilers for systems down to and including micro computers. However, there is not at this time any commercial compiler that implements a full ADA. All

vendors advertise a continued enrichment program and an eventual complete language translator. Telesoft-ADA is a typical example of an available compiler. It is currently available for the VAX/VMS, IBM personal, and the MC68000. For the IBM personal it requires @ 36k bytes of RAM in its present state. Translation time and performance statistics are not readily available. There are also plans by Telesoft to have a translator for the IBM 370/VMS in the near future. These compilers also have an operating system and minimal support environment included. Costs for the IBM/Vm/CMS system is advertised at $11,000 and the VAX/VMS is $9,930. Both will be immediately available after 1 November 1982.

2. Civilian efforts in support of DOD can be seen in the information on United States major ADA efforts provided at TAB 1.

3. Intel Corporation has provided an object based architecture of the Intel 432 for ADA. The underlying address structure is capability base, with access rights included with location. The 432 design provides hardware support of many important features of ADA such as data abstraction, though the 432 is

not an "ADA machine". It gives considerable compile time support to ADA, though it is not specialized for ADA run-time support. The object based architecture does provide a small protection domain, direct support of data abstraction, an enviornment for effective memory management, and should simplify the job of writing compilers for ADA. This machine is presently available. For a complete description of the Intel 432 approach see (ZEIG81).

4. In October, 1981, Intellimac made a corporate commitment to ADA. This decision was based on five months of experience with TeleSoft's ADA compiler. Intellimac has, since that time, used ADA exclusively for contracted and in-house software development. The hardware configuration utilizes the Motorola 68000 microprocessor.

This commitment to ADA has led to the development and release of the first commercial applications software package written in ADA: specifically, a 300-man payroll package. The development of these programs, with their ancillary utilities and support packages, has provided significant insight into the merits of the language.

The programmers of Intellimac, after using the language for nine months, indicate that ADA is the preferred language for commercial software development including maintenance, upgrades, and changes. To those who say ADA is too large, too complex, too limited, too slow, etc., they say "poppycock". Additionally they believe motivated programmers can begin programming within three days and that the speed of the payroll program is an order of magnitude faster than the previous package.

3.4 INTERNATIONAL ADA EFFORTS

International support and interest in ADA is strong. An "ADA Europe" organization has been formed to share and interchange information about the ongoing language development and discoveries. Interchange between the United States and Europe is present and members of the ACM's Sig ADA-Tec reguarly attend European meetings and prepare reports for U.S. consumption. The predominate names in the European ADA community are Alsys, Honeywell-Bull, and Siemans. The academic endeavors appear to be dominated by

the University of Karlsruhe and the University of York. Perhaps the best known project is the University Of York's compiler project. It was funded in November 1979 for 3 years to produce a workbench compiler for ADA. The workbench goal implies particularly good diagnostics and a relationship to the APSE. The objective is a low risk ADA compiler operating under Unix, initially on the PDP-11, and eventually moved to a VAX.

In the AUG-SEP issue of Sigplans SIG ADA-TEC, a complete status report of the York Compiler was given. The compiler, as of yet, does not implement all of the ADA facilities but is nearing a full ADA. It presently has successfully compiled a 6000 line program at speeds of @700 source lines per minute on the VAX 11/780-UNIX. First releases of the compiler will go to various British organizations beginning in the fall of 1982. Total compilation and run-time system size of the ADA translator is @ 66k lines of 'C' code or @ 400k bytes. For a more detailed look consult (WAND82).

In April of 1982 the International Standards Organization established an Experts Group on the Programming

Language ADA. This Organization established the group with affirmitive votes from delegates from around the world. The possibility of an ISO ADA standard is becoming more of a reality ever day. Research and development continues to grow in the European community in the three expected areas of, Military, Commercial, and Academic efforts. For an overview of the projects known to be ongoing in Europe, at this time, see Table 2, European ADA Efforts.

3.5 CONCLUSION

ADA is just now reaching a stage where compilers are being released. This seems to be the point where the language will start to realize its potential. The true value of the language can now be tested as it is delivered to the hands of the people who will use it to produce applications programs. History tells us that many problems will surface and revisions to the language will certainly be made. As the language evolves refinement and solidification will make it more and more reliable and productive. In several articles, of recent weeks, predictions of ADA dominating the world of production languages is expressed almost as a certainty. Some agree that Government support of COBOL helped earn its place as the number one production language in the U.S. It

remains to be seen exactly how far ADA will go. For the interested reader, the probable impact and future of ADA is presented in; (KLIN79),(LEBL81), or (LOVE81). For a concise listing of what I consider pertinent ADA publications please consult Tab 3.

TABLE 1

UNITED STATES ADA EFFORTS

CORPORATION	CONTACT	SCOPE or HOST
Amdahl Corp	John Feiber (408) 746-7052	Full Compiler for IBM 370 INTEL 8086. Used on AMDAHL 470 IBM-370 Compatible
Bell Labs	Charles Wetherell (201) 582-3099	Subset Compiler & Full Proto on VAX/UNIX
Bolt , Beranek & Newman	Bob Thomas (617) 491-1850	Partial Implementation based on Praxis Compiler VAX/PDP11, DEC-20
Burroughs Federal & Special Systems	Terri Payton (215) 648-7268	ADA front end to Diana Burroughs Large Systems
Carnegie-Mellon (SPICE)	Mario Barbacci (412) 578-2578	Subset-Compiler PERQ (Pascal) (DARPA)
Control Data Corporation	Clyde Roby (404) 955-0702	Full ADA with partial enviornment. Cyber-170
Digital Equip. Corporation	Charlie Mitchell (603) 884-8107	Full ADA , VAX/VMS
Florida State University	Ted Baker (901) 644-5452	Full ADA , VAX/VMS For U.S.A.F.
Honeywell Small Systems	Alan Lyman (617) 671-2807	Subset Honeywell Level 6 DPS-6
Intel Corporation	Gary Raetz (503) 642-6103	Full ADA,Linker,Debugger Intel APX-432
Intermetrics Inc.	Mike Ryer (617) 661-1840	Full ADA,debugger PDP-10 and TOPS-20/ for AJPO
" " "	" "	Full ADA IBM 370, PE 8/32 U.S.A.F.
Mills International	(217) 398-1986	Full ADA (eventually) Burroughs B5900,6900,7700

New York University	Gerry Fisher (212)460-7496	Executable semantic model of Language. Vax/VMS
Old Dominion University	Robert Mathis (804)440-3901	Subset written in itself DEC-10
Pr Software	Randall Brukhardt (608)244-6436	Designed tc run on, generate code for micros-8086 ms-dos, 8086 cp/m
Science Applications, INC.	Bruce Lightner (714)454-3811	Full ADA eventually/HP1000 series (Pascal)
Softech	Larry Weissman (617)890-6900	Full ADA (also translator ADA to DEC Pascal) plus support enviornment. For U.S. Army CECCM VAX,PDP
Stanford University	David Luckham (415)497-1242	ADA-M subset-Full ADA in progress. DEC-10,TI990
Telesoft	Ken Bowles (714)457-2700	Full ADA eventually.MC-68000,8086,VAX,IBM-370
Univac Defense Systems	Joe Cross (612)456-3895	ADA subset, no tasking Univac 1100
USAF Armament Lab	CPT J. Bladen (904)882-8264	In house cross-compiler written in Pascal-then to ADA.
USC-ISI	Steve Crocker (213)822-1511	Interpreter for Formal Semantic Definition.For DARPA.
Western Digital	Mary Thorne (714)966-7740	'MicroADA' a subset no Generics,into UCSD Pascal environment.

TABLE 2

EUROPEAN ADA EFFOETS

EACP CII Honeywell Bull /Siemens	J.C.Heliard France (33-3)9181244	Full ADA mini-micros Cii HB level 64 , Siemans 7760
Entwicklunssbuero WERUM	W.Worum (+49)4131-53344	Symbolic Test/Debug System highly retargetable/rehost Siemens 7,xxxx BS 2000
GPP Munich	George Romanski (89)681056	Retargetable Back end for ADA , input is Diana from Karlsruhe Front end.
Int'l Computers LTD	Alan Montgomery (44)782-29681	Systems programs, subset ICL 2900
Olivetti/Danish	Ole Oest 02-872622	Full ADA compiler plus enviornment.Rovsind CR80D Olivetti S6000
OY Softplan AB	Pekka Lahtinen (+358)3137317	Subset for systems sftware development on MPS 10 and Bootstrap on MIKK03
SESA-Deutschland GmbH	Udo Stegen 06-11-717211	Specification and realization of ADA I/O
SPERBER Project; Univ. Karlsruhe	G. Fickenscher (+201)400-6393	Standardization Program system for military apps. Siemens 7,XXXXX
Univ. of Hamburg	Prof, H. Nadel 040-4123-4151	ADA for image processing Apse Implementation, DEC-10,Target German MINI's..
" "	Georg Winterstein (+49)721-608-3005	Front end for full ADA Siemens 7760 Virtual Diana
Univ. of York	Dr. Ian Wand 011-44-904-59861	ADA subset PDP-11,Unix,

TABLE 3

ADA PUBLICATIONS

A Common Language For Computers; Business Week; 23 March 1981; pp84-85.

ADA Test and Evaluation; Intermetrics, Inc; Report IR-663 ; 6 FEB 1981 .

ADA Compiler Validation Capability; Long Range Plan; Softech, Inc; Report 1067-1.1 ; Feb 1980.

ADA Envicrnment Workshop; DOD High Order Language Working Group; 27-29 Nov 1979.

ADA Integrated Enviornment System Specification (DRAFT) Texas Instruments 15 March 1981.

Ada Integrated Enviornment (AIE) Design Rationale: Technical Report (Interim); Intermetrics, Inc. and Massachusetts Computer Associates, Inc.; Report IR-684 ; 13 March 1981.

Ada Integrated Enviornment: Computer Program Development Specification, Part II; Computer Sciences Corporation , Software Engineering Associates, and Verified Computing Services; 15 March 1981.

Ada Integrated Envircnment: Computer Program Development Specification, Part I; Computer Sciences Corporation, Software Engineering Associates, and Verified Computing Services; 15 March 1981.

Ada Integrated Environment: Design Presentation; Texas Instruments Inc.; 14 April 1981.

Ada Integrated Envircnment: Interim Technical Report; CSC; 15 March 1981.

Ada Joint Program Office Information Bulletin: Defense Advanced Research Projects Agency; February 1981.

Ada Language Reference Card: Intermetrics, Inc.; March 1981.

Ada Newsletter Number2; Department of Defense, HOLWG; 20 August 1979. Available in SIGPLAN NOTICES; October 1979; pp.16-35.

Ada PROGRAMMING LANGUAGE ; DOD; Report MIL-STD-1815;
10 December 1980.

ADA Software Environment: Computer Prcgram Development
Specification; Texas Instruments, Inc.; 15 March 1981.

ADA Support System Study: An Initial Discussion Document;
Systems Designers Limited; 31 January 1979.

Ada Support SYSTEM STUDY,Phase I,II,II,IV,15 June 1979,
1 November 1979,1 June 1980, 1 April 1980.

ADA Test and Evaluation Newsletter 1; DOD , HOLWG; 25 May 1979.
Also available in SIGPIAN NOTICES; September 1979; pp. 77-80.

ADA-A Report to the Department of Industry; Department of
Industry; 11 May 1979.

ADA: A History and Prognosis (draft) ; Advanced Computer
Techniques; January 1981.

Computer Program Development Specification for Ada Integrated
Environment: Ada Compiler Phases Type B5; Intermetrics, Inc. and
Massachusetts Computer Association, Inc.; Reprt IR-677:
13 March 1981.

Design Evaluation Report for the ADA Integrated Environment;
Computer Sciences Corporation and Software Engineering
Associates, Inc.; 11 May 1981.

Diana Reference Manual; Institue Fuer Informatick II,
Universitat Karlsruhe and Department cf Computer Science,
Carnegie-Mellon University; Report 1/81; March 1981.

Draft Ada Compiler Validation Implementers' Guide;
Softech,Inc.; Report 1067-2.2; 22 August 1980.

Formal Definition of ADA; interim draft; Honeywell, Inc. and CII
Honeywell Bull; October 1979.

Preliminary ADA Reference Manual; Sigplan Notices, Part A;
June 1979.

Proceedings of the ADA Debut, Washington D.C., 4-5 SEP 1980;
Defense Advanced Research Projects Agency; Report AD-A095 569/0;
September 1980.

Rationale For the Design Of The Green Programming Language; Honeywell , Inc. and CII Honeywell Bull; 15 February 1978.

Requirements for Ada Language Integrated Computer Environments; Preliminary Stoneman; November 1979.

Requirements for High Order Computer Programming Languages; Ironman; 14 January 1977.

Requirements for the Programming Environment for the Common High Order Language; Pebbleman; July 1978

Review of Coral 66 and RTL/2 Features Against ADA; Department of Industry; 18 May 1979.

Set of Sample Problems for DOD High Order Language Program; Green Sclutions; Honeywell, Inc. April 1979.

Standard Compiler Workshop Final Report; Air Force Armament Laboratory; 28 September 1978.

Strawman; DOD , HOLWG; 29 July 1975.

The ADA Language System of the United States Army; Softech; 1981.

Albrecht, Garrison, Graham, Herle, and Krieg-Bruckner; Source-to-Source Translation; Ada to Pascal and Pascal to ADA; Sigplan Notices; November 1980; pp. 183-193.

Amoroso, Wegner , Morris, White, Lopper, Campbell; Report to the High Order Language Working Group (HOLWG); Report AD-A037 634; 14 January 1977.

Archer, J E; An Instructional Subset of the Programming Language ADA; Cornell University; Report TR-79-395, undated.

Arnold, R D; The Nebula Architecture: ADA Issues; ADA letters; May 1981.

Badault, Fisher, Ichbiah, Robert and Wegner; Comments to "Tinman"; 22 July 76.

Baker, F T; A Concurrent Module in ADA; IBM Software Engineering Exchange; October 1980; pp. 18-20.

Bauner, J and Svensson, G; An Implementation and Evaluation of the Real-Time Primitives in the Programming Language ADA, Technical Report; Department of Telecommunication and Computer

Systems, The Royal Institute of Technology (Stockhclm); Report TRITA-CS-8001; April 1980.

Belmont, P A; Type Resolution in ADA; An Implementation Report; Sigplan Notices; November 1980; pp. 57-61.

Belz, F C, Blum, E K, and Heimbigner, D; A Multi-Processing Implementation-Oriented Formal Definition of ADA in SEMANOL; Sigplan Notices; November 1980; pp. 202-212.

Berning,P T; Formal SEMANOL Specifications of ADA; Rome Air Development Center; Report RADC-TR-80-293; September 1980.

Bishop J M; Effective Machine Descriptors for ADA; Sigplan Notices; November 1980; pp. 235-242.

Bishop, A and Johnson, R C; Ada Computer on five boards set to bow; Electronics; 13 January 1981; pp. 39-40.

Bolz, D and Booch, G; Software Engineering with ADA; Department of Astronautics and Computer Science, United States Air Force Academy; March 1981.

Booch, G; Ada promotes software reliability with Pascal-like simplicity; EDN; 7 January 1981; pp. 171-180.

Boute, R T; Simplifying ADA by Removing limitations; Sigplan Notices; October 1980; pp. 27-34.

Brender, R F; The Case Against ADA as an APSE Command Language; Sigplan Notices; February 1980; pp. 27-34.

Brosgol, B M; TCOL-ADA and the "Middle End: of the PQCC ADA Compiler; Sigplan Notices; November 1980; pp.101-112.

Buneman, P, Menten, 1, and Root, D; A Codasyl Interface for Pascal and ADA; Moore School, University of Pennsylvania; August 1980.

Buxton, J N and Druffel, L E; Requirements for an ADA Programming Support Enviornment; Raticnale for Stcneman: Procedings of COMPSAC; 28-30 October 1980.

Carlson, W E; ADA: A Promising Beginning; Computer; June 1981; pp. 13-15.

Carlson, W E; ADA-A Standard Programming Language for Defense Systems; Signal; March 1980; pp. 25-28.

Clapp, J A, Loebenstein, E, and Rhymer, P; A Cost/Benefit Analysis of High Order Language Standardization; MITRE; Report P 78-206;September 1977.

Clarke, L A, Wileden, J C, and Wolf, A I; Nesting in ADA programs is for the Birds; SIGPLAN Notices; November 1980; pp. 139-145.

Cole,S N; ADA Syntax Cross Reference; Sigplan Notices; March 1981.

Cormack, G V; An Algorithm for the Selection of Overloaded Functions in ADA; Sigplan Notices ; February 1981.

Cornhill, D and Gordon, M E; ADA-the latest words in process control; Electronic Design; September 1980.

Dausmann, M, Persch, G, and Winterstein, G; ADA-O Reference Manual, Operating system manual, and users guide; November 1979.

Deremer, F and Pennello, T; Syntax Chart of ADA_compilation; November 1980.

Dijkstra, E W; DOD-I: The Summing UP: Sigplan Notices; July 1978; pp. 21-26.

Donzeau-Gouge, V, Kahn, G, Lang, B, and Krieg-Brueckner, B; On the Formal Definition of ADA; Rivista di Informatica (Italy); January 1980.

Downes, V A, and Goldsack, S J; The Use of the ADA Language for Real-time Programming of a Distributed Sustem; Department of Computing and Control, Imperial College of Science and Technology; 1980.

Druffel, L E; ADA Compiler Validation (DRAFT); DOD; November 1980.

Druffel, L E; ADA- How will it Affect College Offerings?; Interface; September 1979.

Duncan, A G, and Hutchison, J S; Using ADA for Industrial Embedded Microprocesses or Applications; Sigplan Notices; November 1980; pp.26-35.

Eventoff, W, Harvey, D, and Price, R J; The Rendezvous and Monitor Concepts; Is there an Efficiency Difference?; Sigplan Notices; November 1980.

Fairley, R E; ADA Debugging and Testing Support Environments; Sigplan Notices; November 1980.

Firth, R; Universal ADA Language Issue Report Construction KIT; Sigplan Notices ; May 1980; pp.35-36.

Fisher, D A; A Common Programming Language for Defense Analyses; Report P-1191; June 1976.

Fisher, D A; DOD's Common Programming Language Effort; Computer; March 1978.

Fisher, D A; "Woodenman" a Set of Criteria and Needed Characteristics for a Common DOD High Order Programming Language; Institute for Defense Analyses; August 1975.

Fisher, J; Syntactic Error Recovery in the NYU ADA Compiler; ADA Implementor's Newsletter; September 1980.

Galkowske, J T; A Critique of the DOD COmmon Language Effort; Sigplan Notices; June 1980; pp. 15-18.

Galkowske, J T; Modularity And Data Abstraction In ADA; IBM Software Engineering Exchange; October 1980; pp.13-17.

Gann, S O; A Question Of Type; Have You anything to Declare?; Datalink (Great Britain); February 1980.

Gann, S C; Taken to Task(ADA); Datalink (Great Eritain); March 1980.

Gann, S C; You've Got to Get Your Syntax Right ; Datalink(Great Britain); November 1979.

Ganzinger, H and Ripken, K; Operator Identification in ADA; Formal Specification, Complexity, and Concrete Implementation; Sigplan Notices; February 1980; pp. 30-42.

Glass, R L; From Pascal to Pebbleman...and Beyond; Datamation; July 1979; pp.146-150.

Goodall, A; A Criticism of the Select Statement in the ADA Programming Language; November 1979.

Goodenough, J B; The ADA Compiler Validation Capability; Computer; June 1981; pp. 57-64.

Good, D I, Young, W D, and Tripathi, A R; An Evaluation of the Verifiability of ADA; September 1980.

Goos, G and Winterstein, G; Towards a Compiler Front-End for ADA; Sigplan Notices; November 1980.

Gordon, M E and Robinson, W B; Using Preliminary ADA in a Process Control Application; AFIPS Conference Proceedings, 1980 Nationl Computer Conference; pp.597-606.

Groves, L J and Roger, W J; The Design of a Virtual Machine for ADA; Sigplan Notices; November 1980; pp. 223-234.

Habermann, A N and Nassi, I R; Efficient Implementation Of ADA Tasks; Department of Computer Science, Carnegie-Mellon University; Report CMU-CS-80-103; January 1980.

Haridi, S, Bauner, J, and Svensson, G; An Implementation and Empirical Evaluation of the Tasking Facilities in ADA; Sigplan Notices; February 1981; pp. 35-47.

Hibbard, P,M Hisgen, A, Rosenberg, J, Sherman, M; Programming In ADA: Examples; Department of Computer Science, Carnegie-Mellon University; Report CMU-CS-80-149; October 1980.

Hoare, C A R; Subsetting of ADA; draft ; June 1979.

Ichbiah, J d; Ada and the Development of Software Components; Proceedings of 4th International Conference on Software Engineering; September 1979.

Ichbiah , J D, Maddock , R F, and Pyle, I C; Comments on "Ironman"; LTPL-E; Report JI/RM/IP 770606; 6 June 77.

Janas, J M; A comment on "Operator Identification in ADA" by Ganzinger and Ripken; Sigplan Notices; September 1980; pp.39-43.

Johnson, R C; ADA's Modularity sparks interest for civilian uses; Electronics; December 1980.

Johnson, R C; Special Report: ADA, the ultimate language?; Electronics; 10 February 1981; pp.39-40.

Jones, D W; Tasking and Parameters: A Problem Area in ADA;

Sigplan Notices; May 1980; pp.37-40.

Kling R and Scacchi, W; The DOD Common High Order Programming Language Effort; What will the Impact be?; Sigplan Notices; February 1979; pp.29-43.

Knobe, B; Flight Languages; ADA vs HAL/S; Journal for Guidance and Control; January 1981; pp. 35-40.

Krieg-Bruckner, B and Luchham, D C; Anna: Towards a Language for Annotating ADA Programs; SIgplan Notices; November 1980; pp.128-138.

Lamb, D A, Hisgen, A, Rosenberg, J, SHerman, M; THe Charrette ADA Compiler; Department of Computer Science, Carnegie-Mellon University; Report CMU-CS-80-148; October 1980.

Leblanc, R J and Goda, J J; The Impact of ADA on Software Development; Proceedings of SOUTHEASTCON 81.

Locke, D; The ADA Programming Support Environment; IBM SOftware Engineering Exchange; October 1980; pp.21-22.

Loveman, D; ADA resolves the unusual with 'exception' handling; Electronic Design; January 1981.

Loveman, D; ADA: How Big a Difference Will It Make In Software?; Military Electronics/Countermeasures; May 1981.

Loveman, D; The ADA Integrated Environment: An Introduction to the Problem: ADA Letters; March 1981.

Lovengreen, H H and Bjcrner, D; On a Formal Model of the Tasking Concept in ADA; Sigplan Notices; November 1980; pp. 213-222.

Luckham, D C and Polak,W; A Practical Method of Documenting and Verifying ADA Program with Packages; Sigplan Notices; November 1980; pp.113-122.

Luckham, D C and Polak, W; ADA Exceptions: Specification and Proof Techniques: Department of Computer Science, Stanford University; Report STAN-CS-80-789; February 1980.

Luckham, D C and Polak, W; ADA Exception Handling: AN Axiomatice Approach; ACM Transactions on Programming Languages and Systems; April 1980.

Maclaren, L; Evolving Toward ADA in Real Time Systems; Sigplan Notices; November 1980; pp. 146-155.

Maddock, R F and Marks, B L; Towards A PL/1-Based "Ironman" Language;IBM; Report TR.12.168; December 1977.

Mahjoub, A; Some Comments on ADA as a Real-Time Programming Language; Sigplan Notices; February 1981.

Mengarini, B; Macro Facilities in ADA; Sigplan Notices; March 1981; pp.75-82.

Moffat, D.V.; Enumerations in Pascal, ADA and Beyond; Sigplan Notices; February 1981 ; pp. 77-82.

Newton, R ; Some Exception Handling Problems in Language Systems Displaying a Multi-Path Capability; Sigplan Notices; April 1979.

Nicolescu , R; Some SHort COmments on the Definition and the Documentation of the ADA Programming Language; Sigplan Notices; July 1980.

Notkin, D. S.; An Experience with Parrallelism in ADA; Sigplan Notices ; November 1980.

Pennello,T.,Deremer, F.,and Meyers, R.; A simplified Operator identification Scheme for ADA; Sigplan Notices; July 1980.

Persch, G; Overloading in Preliminary ADA; Sigplan Notices; November 1980.

Pyle, I.C.; ADA workshop, University of York, UK, Computers and Digital Techniques 2; December 1979.

Pyle, I.C.; Comment on the SDL/SS ADA Support System Phase 1 Report; June 1979.

Roubine, O, and Heliard, J.C.; Parallel Processing in ADA; Preprint; undated 1979.

Runciman, C; Resolving overloaded expressions in ADA; ADA Implementor's Newsletter; October 1981.

Rymer, J; An ADA Turorial; IBM Software Engineering Exchange; October 1980.

Scarpelli, A.J.; ADA Test and Evaluation ; Air Force Wright Aeronautical Labs; Report AFWAL-TR-80-1024; May 1980.

Schonber, E., and Falk, M; The ADA Tasking Primitives and Their Description in GYVE; New York University; undated.

Schwartz, R. L.and Melliar-Smith, P. M.; On the Suitability of ADA for Artificial Intelligence Applications; SRI International; Report ARO-17127; July 1980.

Sherman, M. S. and Borkan, M.S.; A Flexible Semantic Analyzer for ADA; Sigplan Notices; November 1980.

Sherman, M. Hisgen, A. Lamb, D.A., and Rosenberg, G.; An ADA Code Generator for VAX 11/780 with Unix; Sigplan Notices; November 1980.

Silberschatz, A; On the Sychronization Mechanism of the ADA Language; Sigplan Notices; February 1981.

Stenning, V. Forggatt,T. Gilbert, R. and Thomas E.; The ADA Enviornment : A Perspective; Computer; June 1981.

Stevenson, D.R.; Algorithms for Translating ADA Multitasking; Sigplan Notices; November 1980.

Stroet, J; An Alternative to the Communication Primitives in ADA; Sigplan Notices; December 1980.

Tai, K. and Garrard, K; Comments on the Suggested Implementation of Tasking Facilities in the "Rationale for the Design of the ADA programming Language"; Sigplan Notices; October 1980.

Van Den Bos, J; Comments on ADA Process Communication; Sigplan Notices; June 1980.

Wallis, P.J. and Silverman, B.W.; Efficient Implementation of the ADA Overloading Rules; Information Processing Letters; April 1980.

Wand, I.C.; Systems Implementation Languages and IRONMAN; IBM Thomas J.Watson Research Center; Report RC 7410; October 1978.

Wand , I.C. and Holden, J; Towards a Run-Time System for ADA; Department of Computer Science, University of York; Report YCS-29; December 1979.

Waugh, D.W; ADA As A Design Language; IBM Software Engineering Exchange; October 1980.

Wegner, P; The ADA language and Enviornment; Electro/80 ; High Technology Electronics Exhibition and Convention; May 1980.

Welsh, J and Lister, L; A Comparative Study of Task Communications in ADA; undated.

Whitaker, W.A; ADA- The DOD Common High Order Language; NAECON 1979.

Whitaker, W.A.; Comments on Portions of the ACM Sigplan on the ADA Programming Language not Available in the Proceedings; ADA Implementor's Newsletter; October 1981.

Wirth, N; COmment on the Two Proposals for the Proqramming Language ADA submitted to the Department of Defense; March 1979.

Young, W.D. and Good, D.I; Generics and Verification in ADA; Sigplan Nctices; November 1980.

Young, W. D.and Good, D.I.; Steelman and the Verifiability of ADA; Sigplan Notices; February 1981.

Zeigler, S, ALlegre, N, Johnson, R, Morris, J, and Burns, G; ADA for the Intel 432 Microcomputer; Computer; June 1981; pp. 47-56.

Zuckerman, S; Problems with the Multi-Tasking Facilities in the ADA Programming Language; Defense Communicaticns Engineering Center; Report 16-81; May 1981.

THE FOLLOWING ARE UPDATES SINCE THE OBIGINAL LIST WAS MADE

ADA Language System Bare VAX-11/780 Lcader B5 Specification; SOftech, Inc.; Report 1075-14.2; May 1981.

Jean Ichbiah Assesses ADA and the Future of Microcomputers; Defense Electronics September 1981.

Baker, T.P.; A One Pass Algorithm for Overloading Resolution in ADA; Department of Mathematics and Computer Science, Florida State University; April 1981.

Jessop, W.H.; ADA and Distributed Systems; Department of Computer Science, University of Washington; Report 81-01-06; January 1981.

Rogers, M. A. and Myers, L.M.; AN Adaptation of the ADA language for Machine Generated Compilers; Naval Postgraduate School; Reportl AD-A097 292/7; December 1980.

Shumate, K. C.; ADA-new language that will impact ccmmercial users; Data Management; August 1981.

BIBLIOGRAPHY

(BEN-81) Ben-Ari, Yehudi , A;
Methodology for Modular Use of ADA;
Sigplan Notices ; January 1982.

(BENN82) Bennett, D.A.,Kornman, B.D., Wilson, J.R.;
Hidden Costs In ADA;
SIG ADA-TEC ; January 1982.

(BOS-80) Van Den Bos, J; Comments on ADA Process
Communication; Sigplan Notices; June 1981.

(BROS82) Brosgol, B.M.;
Summary of ADA Lnaguage Changes;
SIG ADA-TEC; January 1982.

(FISH76) Fisher, D.A.; "Woodenman": A set of Criteria
and needed Characteristics for a Common DOD
HOL; Institute for Defense Analyses;
August 1975.

(CARL81) Carlson, W.; ADA : A Promising Beginning;
Computer ; June 1981 ; pp. 13-15.

(GHEZ82) Ghezzi, C. , Jazayeri, M. ;
Programming Language Concepts;
John Wiley and Sons Inc. ; 1982.

(GLAS79a) Glass, R.L.; "Software Reliability at
Boeing Aerospace : Some New Findings" ;
Boeing Company Document D180-25392-1,
September 1979.

(GLAS79b) Glass, R.L.; Real Time Software Debugging and Testing; Boeing Company Document D-180-25249-7,2,3; September 1979.

(GLAS79) Glass, R.L.; From Pascal to Pebbleman...and Beyond ; Datamation ; July 1979 ; pp. 146-150.

(GOOD81) Goodenough , J.B.; The ADA Compiler Validation Capability ; Computer ; June 1981 ; pp. 57-64.

(HOAR81) Hoare, C.A.R.; The Emperor's Old Clothes ; Communications of the ACM ; February 1981; pp.75-83.

(KLIN79) Kling, R. , Scacci, W. ; The DOD Common High Order Programming Language Effort (DOD-1): What will the impacts be? ; Sigplan Notices ,Vol 14, Number 2, 1979.

(LEBL81) Leblanc, R.J. and Goda, J.J.; The Impact of ADA on Software Development; Proceedings of SOUTHEASTCON '81'; April 1981.

(LEDG82) Ledgard, H.F., Singer, A. ; Scaling Down ADA (or towards a standard ADA subset); Communications of the ACM ; February 1982.

(LOVE81) Loveman, D; ADA: How Big a Difference Will it Make In Software?; Military Electronics/ Countermeasures; May 1981.

(RATD79) Rationale for the Design of the ADA Programming Language; Sigplan Notices, Vol 14; June 1979.

(SILB81) Silberschatz, A; On The Synchronization Mechanism of the ADA Language; Sigplan Notices; February 1981.

(USDD75) U.S.DOD; Defense Department Requirements for High Order Programming Languages: Strawman; DOD HOLWG; July 1975.

(USDD77) U.S. DOD; Requirements for High Order Computer Programming Language: Ironman; January 1977.

(USDD76) U.S. DOD; Requirements for High Order Computer Programming Language: Tinman; June 1976.

(USDD80a) U.S. DOD; ADA Programming Language;July 1980.

(USDD80b) U.S. DOD; Military Standard-1815; The ADA Programming Language; December 1980.

(USDD80) U.S.DOD ; Requirements for ADA Programming Support Environments : Stoneman ; February 1980.

(USDD78) U.S.DOD; Defense Department Requirements for High Order Programming Languages: Steelman ; June 1978.

(WAND82) Wand, I.C.; The York Compiler; Department of Computer Science, University of York; August 1982.

(WICK82) Wickmann, B.A.; A Comparison of Pascal and ADA; SIG ADA-TEC; August 1981.

(WEGN80) Wegner, P.; Programming with ADA : An Introduction by Means of Graduated Examples ; Prentice Hall ; 1980.

(WOLF81) Wolfe, M.I. , Babich, W. , Simpson, R.,Thall,R., and Weissman, L. ; The ADA Language System; Computer ; June 1981; pp. 37-45.

(YOUN80) Young, W.D., Good, D.I.; Steelman and The Verifiability of (Preliminary) ADA; Sigplan Notices ; June 1980.

(ZELK78) Zelkowitz, M.V.; Perspectives on Software Engineering; ACM Computing Surveys Vol. 10; June 1979.

AN ADA REVIEW:

A History , Problems and Complaints ,

and the Current Status of the Language

by

MICHAEL IRWIN HODGES

B.A. , University of Pacific , 1975

M.S. , University of Southern California , 1978

AN ABSTRACT OF A MASTER'S REPORT

Submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan , Kansas

1983

ABSTRACT

This report is a review of the programming language ADA. The report covers three areas; the history, the complaints, and the current status of the language. Chapter One, the History, introduces the reasons why the Department Of Defense recognized they had a computer language problem and why after some analysis perceived a need for a new High Order Computer Language.The subsequent language development is presented.

Chapter Two, complaints from the computer community, requires some knowledge of language design and an in depth knowledge of some computer concepts. The complaints section presents general information about the features; the specific complaint and in some cases my personal comment pertaining to the complaint. The complaints presented center around the areas of,Language Design, Major Complaints, and Minor Problems. Language Design is presented in terms of size, complexity, uniformity, and portability of the programming language. Major Complaints, involve the semantic features of tasking, pragmas, and exceptions.

Chapter Three presents an overview of the current ADA status. Three major sectors of development are presented. They include ; the Department of Defense, Commercial efforts, and International ADA development. Provided as tabular data, and as a concise reference, are; all known U.S. and International ADA efforts as well as a list of ADA publications.