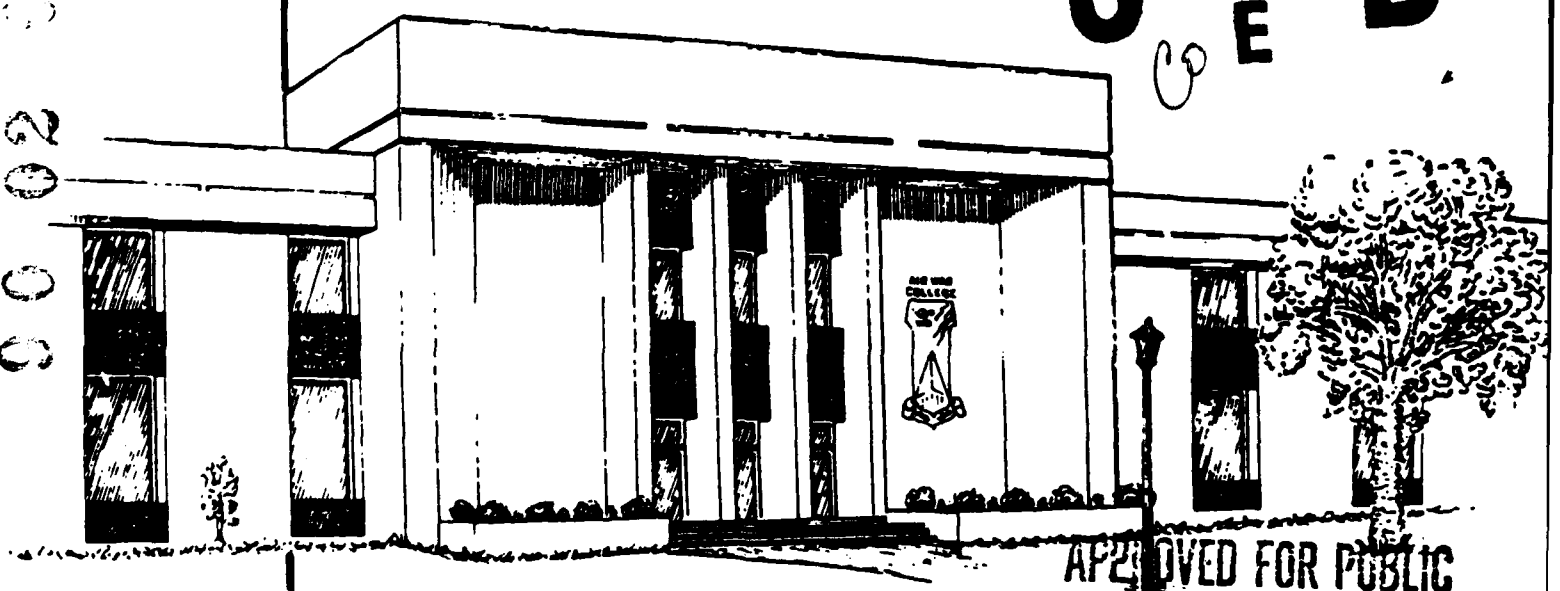
AIR WAR COLLEGE

RESEARCH REPORT

ADA, THE NEW DOD WEAPON SYSTEM COMPUTER LANGUAGE - PANACEA OR CALAMITY

LT COL NICHOLAS J. BABIAK

1989

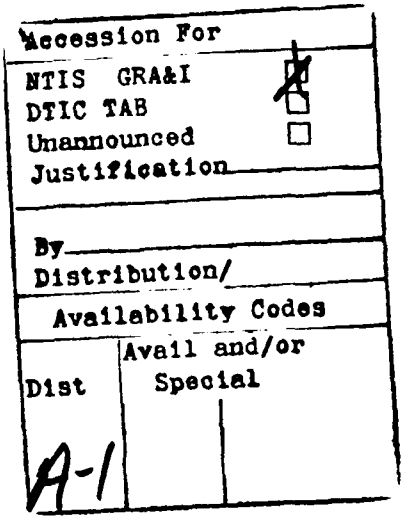


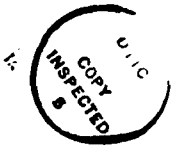
AIR UNIVERSITY
UNITED STATES AIR FORCE
MAXWELL AIR FORCE BASE, ALABAMA

DISCLAIMER

This study represents the views of the author and does not necessarily reflect the official opinion of the Air War College or the Department of the Air Force. In accordance with Air Force regulation 110-8, it is not copyrighted but is the property of the United States government.

Loan copies of this document may be obtained through the interlibrary loan desk of Air University Library, Maxwell Air Force Base, Alabama 36112-5564 (Telephone: [205] 293-7223 or AUTOVON 875-7223).





AIR WAR COLLEGE
AIR UNIVERSITY

ADA, THE NEW DOD WEAPON SYSTEM COMPUTER LANGUAGE - PANACEA OR CALAMITY

by

Nicholas J. Babiak
Lieutenant Colonel, USAF

A DEFENSE ANALYTICAL STUDY SUBMITTED TO THE FACULTY

IN

FULFILLMENT OF THE CURRICULUM

REQUIREMENT

Advisor: Mr. Robert O. Dahl

MAXWELL AIR FORCE BASE, ALABAMA

May 1989

//

EXECUTIVE SUMMARY

TITLE: Ada, The New DoD Weapon System Computer Language - Panacea or Calamity

AUTHOR: Nicholas J. Babiak, Lieutenant Colonel, USAF

The number of computers embedded in DoD weapon systems is escalating at unprecedented rates. Current emphasis has shifted to software as the preeminent focus for new weapon systems because of software's potential to increase the capability of the weapon system and the high costs incurred during development and support phases of a weapon system's life. The problem in this study was that of determining the extent to which Ada has achieved its intended purpose of functioning as a standard programming language for DoD weapon systems. Consequently, this study examined whether this programming language has corrected the perceived software problems of past software languages or whether Ada use has repeated the problems of past computer languages as well as caused newer problems that nullify achievement of its selection objectives. The studies hypotheses looks to Ada's intended purpose and whether Ada can eventually resolve past software language problems in the areas of cost, reliability, system capability, maintainability, and weapon selection. The study concluded with an assessment that Ada's original intentions have been successful, however the study falls short of endorsing all of Ada's claims. Recommendations are given to facilitate a wider acceptance of Ada.

BIOGRAPHICAL SKETCH

Lieutenant Colonel Nicholas J. Babiak (M.A. in Business Management, Central Michigan University; B.S. in Electrical Engineering [Computer Design] University of Buffalo; post graduate work in computer technology) has been active in working the many issues of computer technology for weapon systems. He has recently held the position of Chief, Emerging Technologies in the Office of the Special Assistant for Reliability and Maintainability, responsible for increasing Air Force war fighting capability by accelerating new reliability and maintainability (R&M) technologies into developmental and fielded systems. He has held various positions working in research and advanced technologies. He served as Deputy Director of the DoD Very High Speed Integrated Circuits (VHSIC) Research and Development program and Principal Advisor, Embedded Computer Resources at the Air Staff. He is a graduate of Armed Forces Staff College and Squadron Officer's School. Lieutenant Colonel Babiak is a graduate of the Air War College, class of 1989.

TABLE OF CONTENTS

CHAPTER ONE

INTRODUCTION

The Rise of Digital Electronics and Weapon Systems

The past few decades have felt the impact of digital electronics in the form of the computer on virtually every walk of life. Computers have made possible the seemingly impossible. From such epic achievements as placing a man on the moon to monitoring anesthesia in hospitals, computers account for a myriad of everyday conveniences. The world is increasingly becoming dependent on the computer to solve its problems.

The pace of computer technology has spiralled in just the latter part of the last decade. About 4000 years ago, civilizations developed sophisticated numbering systems, but mankind did not dispose of the first computer hardware until an Englishman named Charles Babbage invented his Analytical Engine to perform a wide range of computational tasks from a sequence of instructions we now call *software*.

Recent gains in computer power and versatility stem from a tiny technological miracle called the microprocessor, scarcely two decades old. With the advent of the microprocessor, and as a result of seemingly endless advancements in the development of the integrated circuits, computers now impact virtually every major weapon system in the U.S. modern military arsenal.

The utilization of computers in fighter aircraft has increased at a staggering rate. Through most of the Vietnam War, F-4s contained no digital computers and no software. With the introduction of the F-16A in 1981, com-

puters became commonplace in weapon systems. The early F-16 had seven computer systems with fifty digital processors and 135,000 lines of software code. In contrast, the new F-16D has fifteen computer systems with 300 digital processors and 236,000 lines of software code. (9:49)

Computers in fighter aircraft and other weapon systems play a decisive role in converting aircraft into modern fighting machines. Computers embedded in these fighter aircraft control everything from electronic warfare systems to flight control systems. How well these computers perform their job determines the fate of the crew member and the mission, from takeoff through combat to returning to the home base.

As Dr. Edith Martin, a former Undersecretary for Defense (Research and Advanced Technology), wrote:

> Our potential adversaries, mainly the Soviets, are numerically superior, technologically sophisticated, well equipped, and prepared. We decided to base our defense strategy on superior technology rather than match those adversaries one-for-one in equipment and manpower. . . almost every defense system fielded today contains a computer and has software performing mission-critical functions. The future success of our forces on the battlefield, should conflict arise, will depend on the maturity of our computer technology and its applications. (36:3)

With our growing dependency on computers in weapon systems, one can foresee that these machines will eventually effect our national security. In a report completed by the Office of the Secretary of Defense in 1982, one finds the following sobering words: "The military power of the United States is inextricably tied to the programmable digital computer." It drew the corollary conclusion that software problems "can make our future military systems fail in ways that could be disastrous for our national security." (9:49)

A few years ago much of the attention in designing and developing the application of computers in weapon systems were centered on the hardware

element (omputer systems. However, in the recent past, the impact of software on these systems is drawing much of the attention. It stands to reason that software actually makes or breaks the capability of the system, for, without software, the embedded computer would only take up room in the aircraft and be of no use.

Analysis Problem and Purpose

The number of computers embedded in DoD weapon systems is escalating at unprecedented rates. In weapon systems of the recent past, hardware was the driving force in system capability, cost, selection, reliability, and maintainability. Currently emphasis shifted to software as the preeminent focus for new weapon systems because of software's potential to increase the capability of the weapon system and the high costs incurred during development and support phases of a weapon system's life. However, Ada is still a relatively new and controversial programming language. The problem under consideration in this study is that of determining the extent to which Ada has achieved its intended purpose of functioning as a standard programming language for the creation of large-scale embedded computer system programs for DoD weapon systems. Consequently, this study will examine the development of Ada since its adoption to the present and actual performance of Ada in use today as well as in developing future programs for weapon systems to determine whether this programming language has corrected the perceived software problems of past software languages or whether Ada use has repeated the problems of past computer languages as well as caused newer problems that nullify achievement of its selection objectives.

Hypotheses

The Department of Defense embarked upon a firm direction to instill Ada as a cornerstone to solving the "software problem." Since the infusion of computers into U.S. weapon systems, controlling software costs, software timeliness, and reliability have been rout with less than desirable results. Solutions to past software problems have not worked. This study has two major hypotheses:

H_1 = Ada has achieved its intended purpose of functioning as a standard programming language for the creation of large-scale embedded computer system programs for DoD weapon systems.

H_2 = With the proper degree of software engineering, Ada, used as a standard computer programing language for weapon system software development, can eventually resolve past software language problems in the areas of cost, reliability, system capability, maintainability, and weapon selection.

Definition of Terms

Due to the number of technical terms distinctive to a discussion of Ada, a glossary of such terms has been provided at page 68. However, the following terms are important enough to be listed in this section:

- Embedded computer - A computer that is integral to a weapon sys-

tem. An example would be the Central Computer on the F-15 weapon system. One function of the Central Computer is that it accepts data from various sensors on the F-15 (air data computer, radar, etc.), makes the required computations and displays the situation to the pilot (heads-up display).

- Mainframe - A physically large computer usually capable of handling many users simultaneously. Before the 1970s, the Mainframe computer was prevalent in government and industry.
- PC - Personal computer - A computer small enough and inexpensive enough to be used in the home.
- Embellish - When used in the context of this paper, it indicates that a contractor accepts the use of Ada along with Ada's support environment, Ada's emphasis on software engineering, and Ada's management philosophies.
- Stand alone - A stand alone computer is a computer whose input and output are not dependant on another system. A PC would be considered a stand alone.

Delimitations of the Study

The major focus of the study is on weapon system development in lieu of other computer based systems such as personnel, finance, logistics support data systems, etc. The study effort assumes that the facts, figures, and data derived from the literature are correct. However, on occasion it was difficult to separate the empirical data as factual or adjusted with political overtones. Ada is a DoD mandated software language that is applicable to all services. Space precludes a detailed examination of Ada's applicability to all services. Therefore, only examples from the Air Force are provided. Since the author

could not talk to everyone using Ada, the study's research was limited to published literature, personal experiences, and personal contacts.

Importance of Software Selection

Factors

Software plays a major role in current weapon systems. The "smarts" of smart weapons are provided by software. Software is crucial to intelligence, communications, command, and control. Software enables computerized systems for logistics, personnel, and finance. The chief "military software problem" is that we cannot get enough software, soon enough, reliable enough, and cheap enough to meet the demands of weapon systems designers and users. Software provides a major component of our war-fighting capability; as such, *proper* software selection is of paramount importance. (33:6)

Software selection factors have been envisioned in a number of ways: cost, schedule, reliability, and capability. The factor that most affected the Department of Defense in the past and was most visible was *cost*.

The cost associated with software can be a two edged sword. On one hand, it is easier and less costly to modify software than it is to modify hardware, as evidenced by the Air Force's experience on the F-111 program. The Air Force upgraded the avionics of the F-111 A/E aircraft by altering their analog computers; the Air Force also upgraded the avionics of the F-111D/F aircraft to obtain the same new capabilities by altering the software in their digital computers. The hardware changes cost fifty times as much as the software changes and took three times as long to complete. In another instance, software changes that improved the accuracy of 550 deployed Minute-

man III Intercontinental Balistic Missiles cost "only $4 million, a fraction of what the corresponding physical [hardware] modification might have cost." (9:50)

On the other hand software costs rise, and, in many circumstances, they are hidden until later in the support phases of the system. Projections indicate that it costs $85 million to develop the software for an F-16D. It costs another $250 million to maintain that software - rectifying its errors, keeping it in shape, updating it - over its anticipated operational lifetime. (9:49)

Looking at the bigger picture, the Department of Defense spends $10 billion a year on software and anticipates the need to triple such spending by 1990. At $3 billion a year, software spending accounts for nearly four percent of the total Air Force budget and is expected to consume ten percent of the budget by 1990. (9:46)

Dr. Edward Lieblein, the former director of computer software and systems in the Office of the Secretary of Defense stated that by 1992:

> Costs for developing, evolving, and maintaining defense software will have grown to become a principal factor in the determination of U.S. defense capabilities. (39:10)

One must keep in mind that the cost of software is but one selection factor. It appears that software timeliness and reliability may be even more critical factors today than the cost of software. Software development cycles are long relative to the development of hardware, relatively unpredictable, and come at the end of a system's development cycle. Therefore, the development of software frequently encounters delays that impact the eventual use and operational capability of a new weapon system. Additionally software developments usually encounter design flaws which critically affect the relia-

bility of the system's capability. While there are a number of selection factors for weapon systems software, most eventually bear directly or indirectly on the cost factor. Clearly a misjudgment of one or all of these selection factors can cause a software problem that will cause poor or unacceptable weapon system performance. (33:7)

Throughout the literature search, interviews, and the personal experience of the author, the preponderance of opinions indicated that the Department of Defense perceived that previous software selected for weapons development represented a "software problem" which required a corrective solution.

Ada as a Solution to the Software Problem

The cornerstone of the solution, as *envisioned* by the Department of Defense, was a computer programming language standard called Ada. Ada was a programming language for the programming of computers embedded within larger systems. Computers internal to aircraft, ships, radars, or command and control systems are used in different ways than in business or data processing applications. Computers in such embedded computer systems typically interface with human operators and external devices in real time, as events are occurring. They read signals from sensors and send commands to electrical and electromechanical devices. Commercial systems such as process control and data communications have similar characteristics. Ironically, even though Ada was designed for embedded systems, the first Ada software product was a stand alone payroll and inventory system for a truck manufacturer. (42:1)

Ada, as a language for constructing large programs to be used in embedded computer systems, included systems such as aircraft or missiles, command and control systems, and computer-controlled radars or weapons that are typically constructed by large teams of programmers, take several years to develop, and have lifetimes spanning decades. Throughout development, the programs are updated, corrected, and modified. Ada was designed to meet the requirements stated in a language specification that established the necessary characteristics of a language for the intended purpose of creating programs for embedded computer systems unlike all previous languages. (42:vii)

Ada was formally tasked to the services as the Department of Defense standard programming language by the following: (33:16)

- June 1983, memorandum from Dr. Richard DeLauer, Undersecretary of Defense (R&E), mandated the use of Ada on all new Department of Defense mission-critical computer procurements entering concept definition after 1 January 1984 or entering full-scale development after 1 July 1984.
- December 1985, Mr. Don Hicks, Undersecretary of Defense (R&E), reaffirmed the mandate to use Ada.
- November 1986, Mr. Kasper Weinberger, Secretary of Defense, reaffirmed the mandate to use Ada.
- April 1987, DODD 3405.1, "Language Policy," stating that lifecycle cost is the criterion.
- March 1987, DODD 3405.2, "Ada in Weapon Systems," formalized the Ada Executive Official and the Waiver Control Officer.

Secretary Weinberger stated at a major conference two years ago:

> What we need now [in the defense industry] is a vast increase in our computational capabilities. That's why it's so important we have standards such as Ada to guide us in that direction. (57:31)

Since Ada was selected as the *DoD-wide* computer programming language for weapon systems computer program development and since Ada has been employed for a number of years, it is important to critically examine the extent to which Ada has achieved its intended purpose of functioning as a standard, sophisticated programming language to create programs for embedded computer systems in DoD's weapon systems.

Study Procedures

The appropriate literature was surveyed, personal experience was utilized, and interviews held from which relevant data was extracted to form the basis of this study. Chapter II unravels some of the computer mystery and establishes a common framework to discuss the subject of Ada. In Chapter III the examination of some historical aspects of Ada development and use are provided to give a framework for the analysis. Chapter IV examines the Ada program from two viewpoints. The first part of the chapter discusses the perceived benefits of the Ada program. The second part summarizes those issues that have caused negative views of the Ada program. The underlying causes of the Ada controversy are reviewed in Chapter V. Chapter VI places the controversy into perspective as it relates to the intended purpose of Ada. A summarization of the study, the conclusions reached, and some recommendations constitute Chapter VII.

CHAPTER TWO

THE COMPUTER'S IMPACT ON TODAY'S WEAPON SYSTEMS

Computers have infiltrated and revolutionized virtually every aspect of modern warfare, from surveillance and weapon systems to communications, navigation, and battle field management. Yet their composition and capabilities are still a mystery to many who wear the uniform. Therefore, the following discussion will unravel some of the computer mystery and establish a common framework to discuss the subject of Ada.

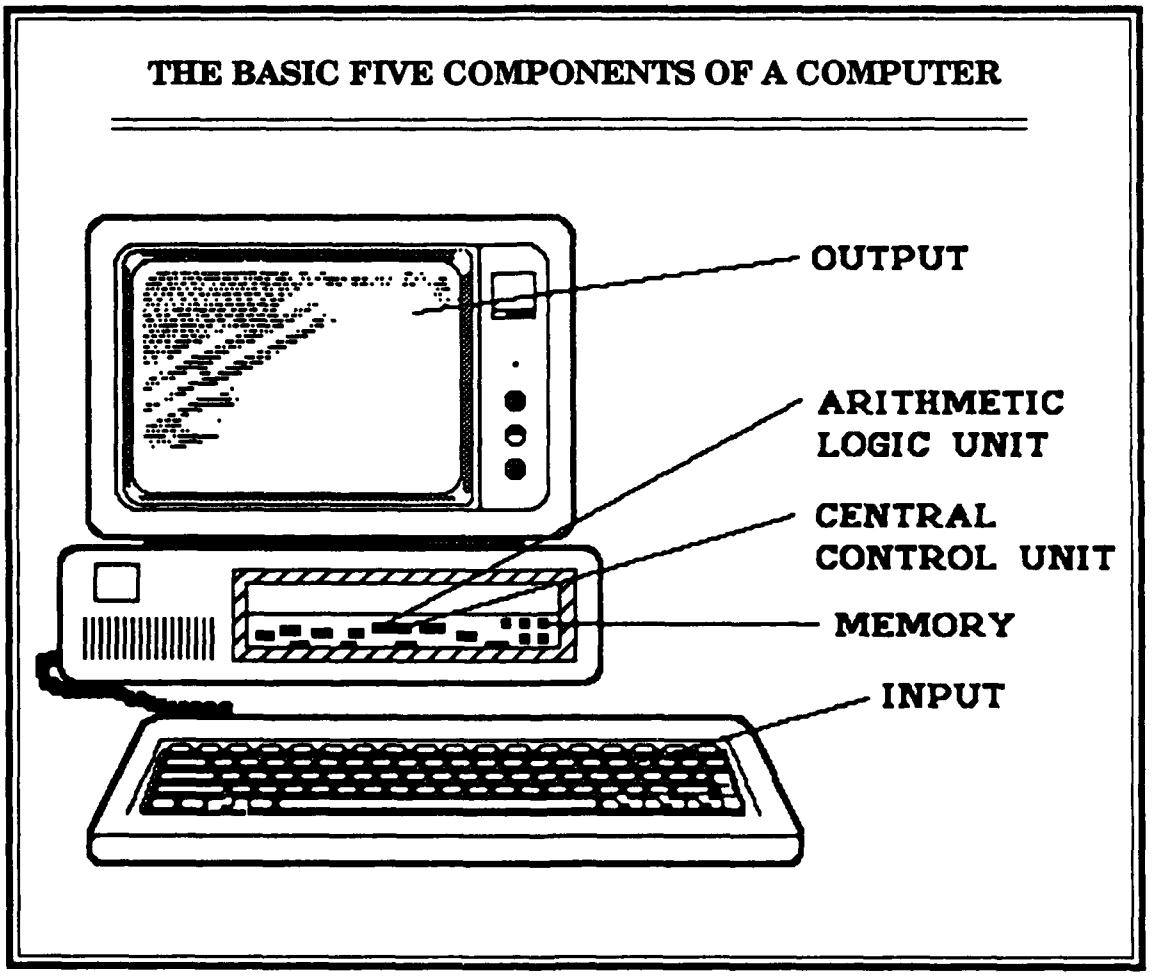
Computer Hardware

In 1945 a Hungarian-born mathematician, John von Neumann, laid out five key components of the computer. These five components have been the fundamental structure of digital computers for many years and have formed the basis of many of our modern day weapon system computers. His architecture called for a:

- Central arithmetic logic unit - This is the part of a computer processing element that performs arithmetic operations such as addition and logical operations.
- Central control unit - This is the part of the computer that is used to orchestrate the operations.
- Memory - There are many types of memories but they all perform the function of storing the program and data.
- Input unit - The most familiar type of input unit is a keyboard. How-

ever many weapon systems may use radar, radio communications, flight control sensors, or any device that takes information and inputs it for computational reasons.

• Output unit - The most familiar type of output is the cathode ray tube (CRT) such as the ones used on today's personal computers.



These five components were the building blocks that guided the design of the early mainframes and still guide many of the computers of today. A more familiar term today is the central processing unit (CPU), usually considered the heart of the computer. The CPU is that part of the computer that interprets and executes the instructions. It contains an arithmetic logic unit, a control unit, and usually a small amount of memory.

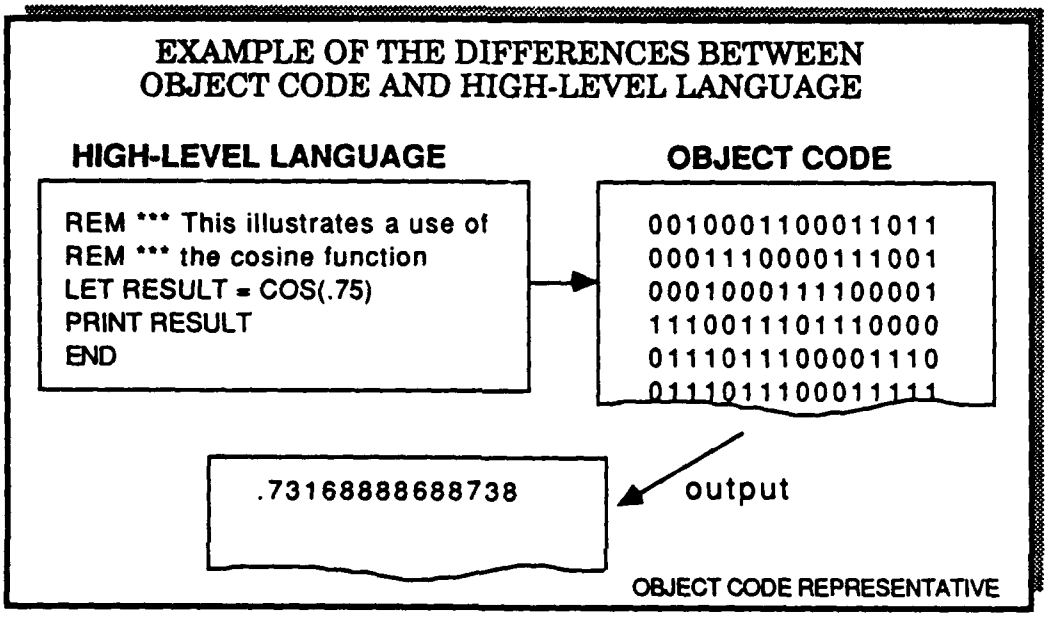
Many contemporary computers still use these five basic components. However, elements of the von Neumann architecture may be combined on one integrated circuit, thus making delineation of these components difficult. Many other architectures have arisen since the 1940's, such as vector processors, parallel processors, etc., but these five elements are still present in one shape or another in most of our weapon system computers.

The physical apparatus of a computer system is called hardware. Computer sizes and shapes vary dramatically from one weapon system to another. The computers that run the War Room of the North American Air Defense Command are extremely large and fill rooms with equipment. In contrast, computers on an F-16 may be only about the size of a typical bread box or may be as small as a finger nail. No matter what the size, shape, or architecture, all of these computers require software to enable the computer to perform work.

Computer Software

Since software is the basis for the remainder of this paper, a description of its elements is appropriate. Software, or a computer program, is essentially a set of instructions which describe the functions which the computer is to perform. These instructions are usually stored in the computer's memory as a sequence of binary codes. Binary codes are a set of numbers which have the value of a one or a zero and are called "object codes." The combination of these ones and zeros make up the instructions or data that the computer uses to perform its function. For a program of any size, the generation of these codes directly by the programmer is a largely impossible task. Hence, high-level programming languages are used to express the

required functions in a notation which is oriented towards the problem that is to be solved rather than towards the machine which is to be used. (58:13)



The basic tool in the translation of a high-level programming language into the corresponding machine code (object code) is performed automatically by software called a compiler. Once this translation is complete, the final program can then be executed. (58:14)

In the early days of high-level programming languages - essentially the period of so-called first-and second-generation hardware - computers ran one program at a time and the principal tool was the compiler. To use a language, a programmer wrote a program, punched it into cards, read the cards into the computer, then attempted to compile and run the program. If there were no errors, the program would operate. (23:124)

DIFFERENT EXAMPLES OF HIGH-LEVEL PROGRAMMING LANGUAGES

```
10    SUM = 0.0
20    DO 30 I = 1,LAST
30    SUM = SUM + A(I)
40    AVER = SUM / LAST
```

FORTRAN

This program is written in FORTRAN (formula translator). The program computes a numerical average.

```
PRINT-AMOUNT-ON-CHECK
      MOVE EMPLOYEE-AMOUNT-TO-BE-PAID
            TO CHECK-DOLLAR-AND-CENTS-AMOUNT
      MOVE EMPLOYEE-AMOUNT-TO-BE-PAID
            TO WORK-TOTAL-AMOUNT
      MOVE WORK-DOLLAR-AMOUNT
            TO CHECK-DOLLAR-AMOUNT
```

COBOL

This program is written in COBOL (common business oriented language). The program is part of a check-writing program.

```
10 INPUT "What is your name?";N$
20 PRINT "Hello," ;N$
30 END
```

BASIC

This program is written in BASIC (beginners all-purpose symbolic instruction code). The program prints out your name.

```
VAR
      message:string;
BEGIN
      message: = `I am
      a program,' ;
      write1n(message);
```

Pascal

This program is written in Pascal (named for the French mathematician Blaise Pascal). The program prints out "I am a program."

With the introduction of so-called third-generation hardware, computers and programs became much more complex. Features such as multiprogramming, time sharing, remote job entry, background and foreground programs executing simultaneously, real-time operations, etc. became commonplace. The need for specialized software - independent program tools - to organize computer operations and to assist in the development, test, running, and post deployment support of application programs became a necessity. These software tools (support software) usually consisted of linkers, editors, loaders, debuggers, and utility routines. (23:124-125)

Historically, support software was not written by language designers. Development of support software was usually left to individual programmers who designed and built special software routines for specific computers as a result of operational needs. The routines were often developed in an unstructured manner, long after the language was in use. Thus, the results were not always good. Software support developed individually for specific user needs led to duplication of effort and often to "reinventing the wheel." This also led to software that operated on only one computer and programmers that were able to program only that computer. (23:125)

Even in today's computer state of the art, a good many support software routines are still primarily ad hoc in nature; they are often independently developed for a particular computer. This leads to dependence on one type of hardware, one operating system, and programs that cannot easily be moved from one computer to another without extensive modification, which in turn results in programming errors, long delays, and high costs. (23:125)

Weapon system dependency on software

The evolutional growth of computers impact the capabilities of today's modern Air Force fighter and bomber aircraft to fly and fight. When airframes such as the F-111 and F-15 were first introduced, they contained computers that integrated many functions into a central computer avionic architecture. The F-111 system contained two computers; one was the weapons computer, the other was dedicated to navigation. Since most of the avionics were analog, the signals had to be converted to digital in order to communicate with the computers. The F-15 had a main central computer which talked to the other avionics via a digital "bus." As computers became more prolific, as on the F-15, computers would talk to other computers and exchange information in a federated architecture. Such an integrated avionic approach is planned for the Advanced Tactical Fighter which employs literally hundreds of computers.

Its easy to see that computers are here to stay on our airborne weapon systems. As these computers multiply so will their software. Richard Behel from TRW's Systems Development Division in Huntsville, Alabama, states:

> Over the last 30 years we have seen tremendous growth in hardware capabilities. For the same cost that used to get you an 8-bit microcomputer with 8,000 words of memory, you can now have a 2 [million instructions per second] MIPS processor and a million words of memory. The problem is that instead of being faced with writing a couple of thousand lines of code, you're carrying a million lines of code. The most recent upgrades of the F-15 are carrying a million lines of code. When you look at the ATF, your carrying 2 or 4 million lines of code. (1:65-66)

Both our fighter and bomber aircraft are increasing their consumption of software. For example, the B-1A bomber of ten years ago embodied 500,000 lines of software code; this has grown to approximately 1,200,000

lines of software code in today's B-1B. (9:46)

Software's impact on mission performance is also growing. Software's responsibilities have grown from weapons release and navigation and has moved from controlling the flight characteries of the aircraft to monitoring and controlling engine performance.

The F-16 Flight Control System is an example of the impact of today's software on mission performance. The F-16 flight control system is a multi-redundant, fly-by-wire system controlled by the pilot and managed by the flight control computers and its ensuing software. The system controls the flight surfaces and relies on the inputs of the pilot, and then optimizes the flight path of the aircraft. One of the functions the flight control system is preventing the overstressing of the aircraft by limiting the "G" forces. It modifies the aircraft's turning capability according to aircraft weight, configuration, and true airspeed. The system is highly dependent on the reliability and speed of the software to perform these functions. An error in the software can cause mission abortment and a possible fatality. (36:3,4)

Since our aircraft systems will become more dependant on software, a rational approach to software development and support needs to be pursued. The Department of Defense places the use of Ada as a cornerstone of this rational approach which is discussed in the next chapter.

CHAPTER THREE

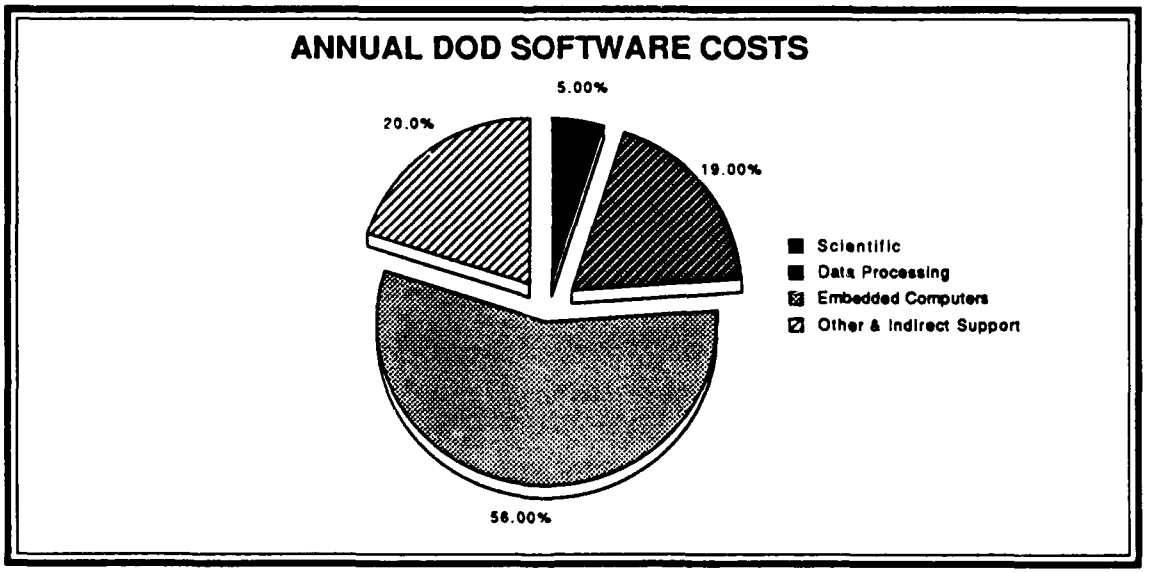
DEVELOPMENT OF ADA

The purpose of this chapter is to give the reader a fundamental understanding of the background of the development of Ada. This review is given to demonstrate the vast amount of work that has gone into the formalization of the present DoD policy.

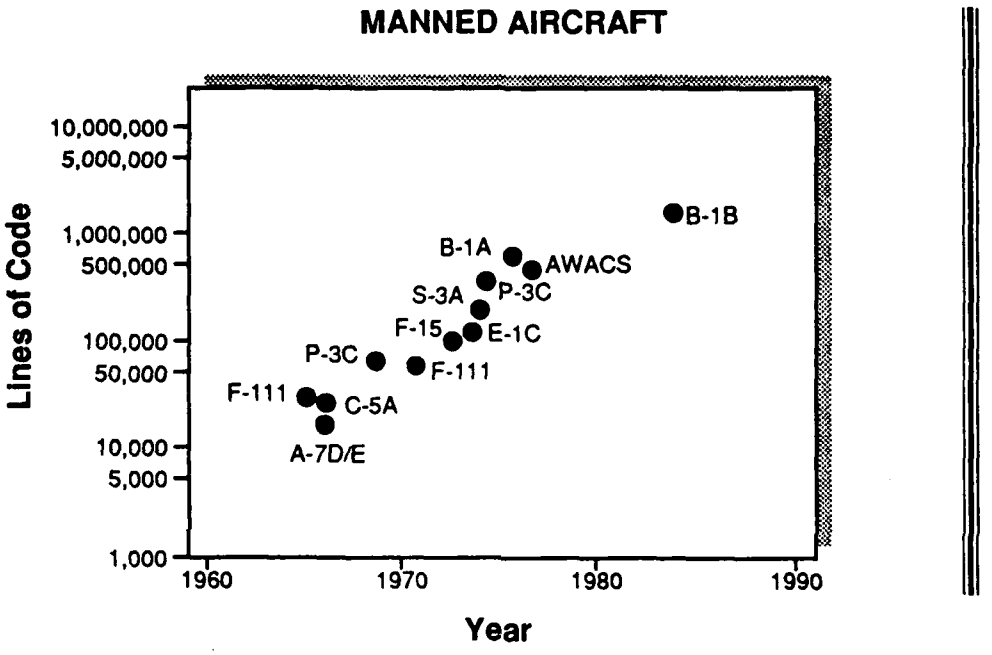
Pre-Ada Software Problem Surfaces

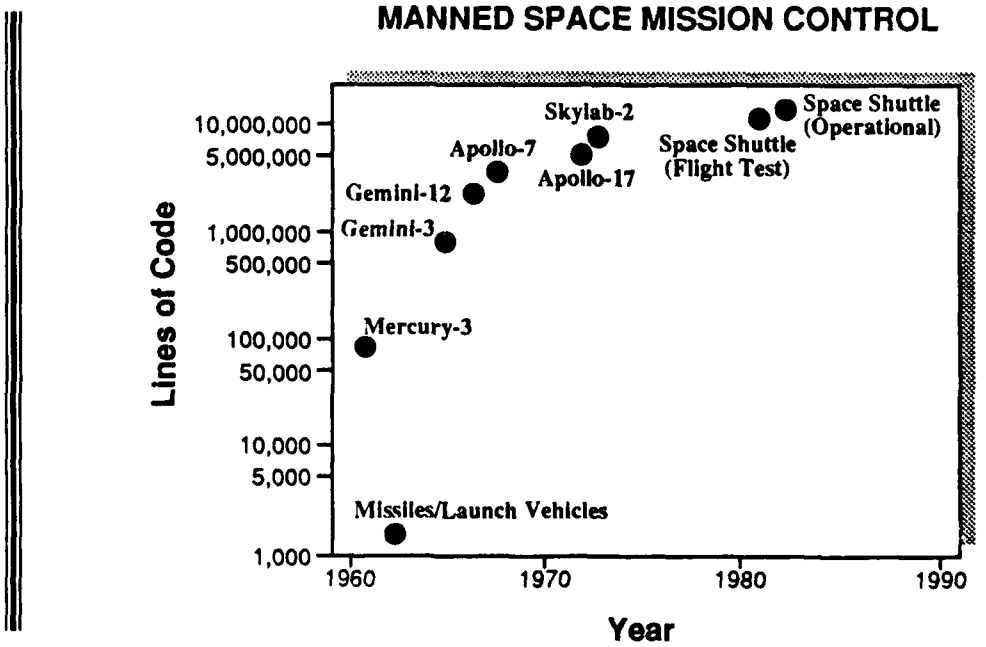
In the early 1970's, the high cost of development and support of computer systems fostered a number of studies. These studies revealed that there was an apparent serious cost problem in the development and support of DoD computer systems. The studies further revealed that most of the cost was related to embedded computer systems.

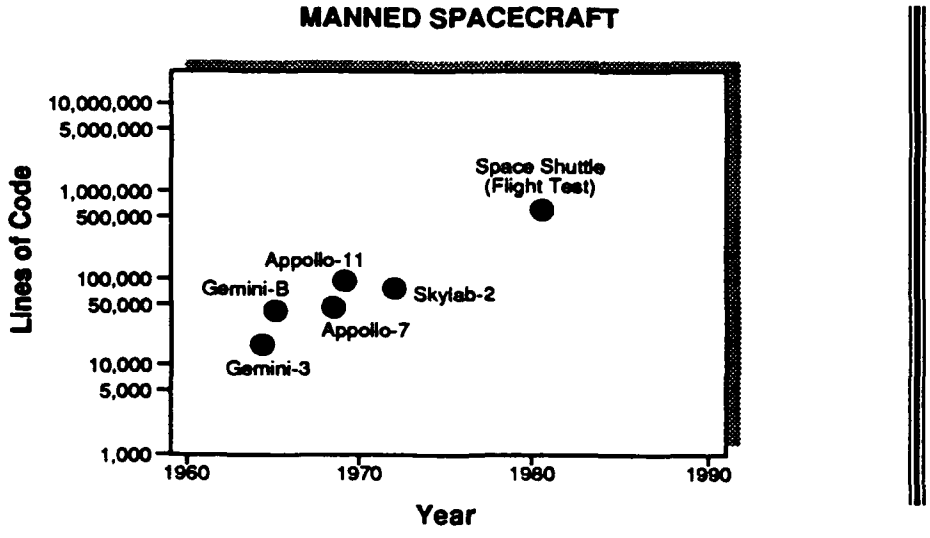
These studies revealed that the DoD spent $3 billion for software in 1973. They concluded that the majority of these costs were not incurred for

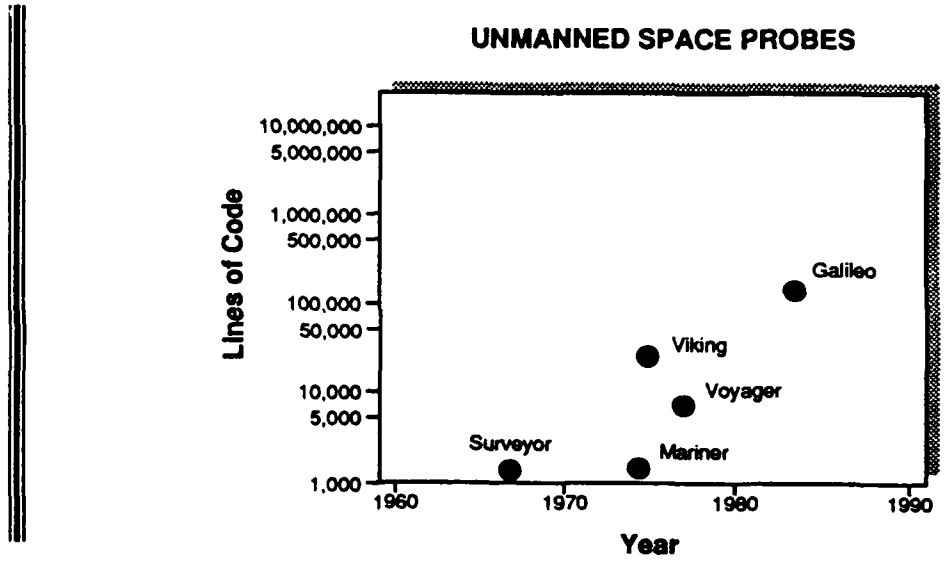


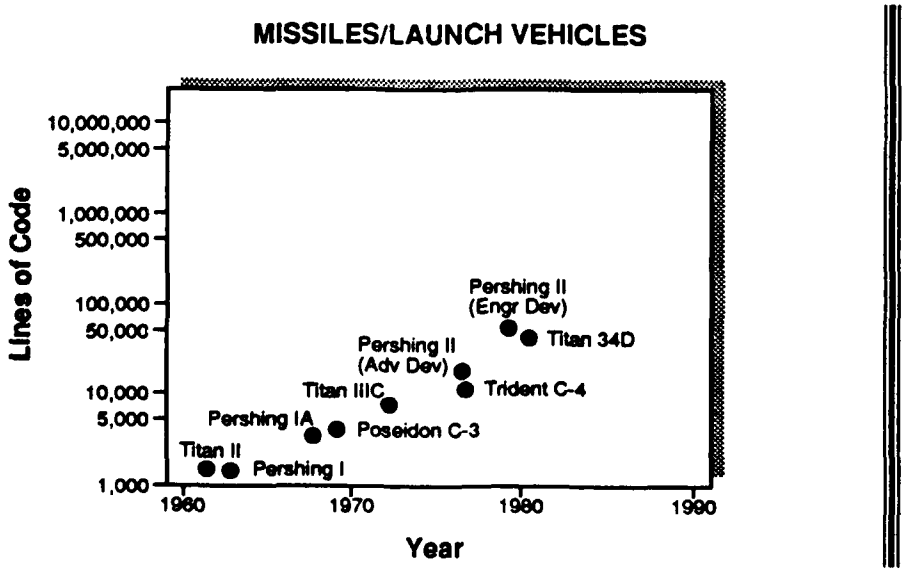
the development of new systems, but rather for supporting and maintaining existing systems. It was also revealed that over 200 models of computers and over 450 general-purpose programming languages and dialects were being used for computer systems. The amount of software (lines of code) was also growing as time progressed. (42:10)






MANNED SPACECRAFT
Lines of Code
10,000,000
5,000,000
1,000,000
500,000
100,000
50,000
10,000
5,000
1,000
Space Shuttle
(Flight Test)
Apollo-11
Gemini-B
Skylab-2
Apollo-7
Gemini-3
1960
1970
1980
1990
Year

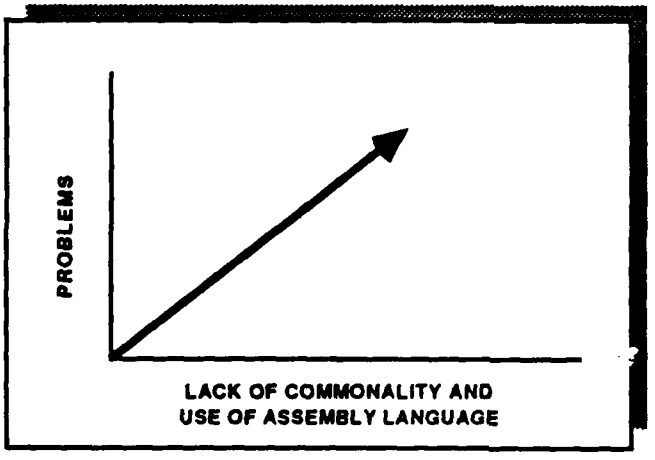

UNMANNED SPACE PROBES
Lines of Code
10,000,000
5,000,000
1,000,000
500,000
100,000
50,000
10,000
5,000
1,000
Galileo
Viking
Voyager
Surveyor
Mariner
1960
1970
1980
1990
Year


MISSILES/LAUNCH VEHICLES
Lines of Code
10,000,000
5,000,000
1,000,000
500,000
100,000
50,000
10,000
5,000
1,000
Pershing II
(Engr Dev)
Titan 34D
Pershing II
(Adv Dev)
Titan IIIC
Trident C-4
Pershing IA
Poseidon C-3
Titan II
Pershing I
1960
1970
1980
1990
Year

During the 1970's, software for embedded computer applications was primarily written in assembly language. Software was also being written using higher-level languages but these programs still contained a high proportion of assembly language. This was done to overcome deficiencies and accomplish functions not amenable to high-level implementation. Computer memory and speed were at a premium in those days and the use of assembly language was the best means to work with real time applications. The lack of commonalty and use of assembly language made the development of new software difficult and created an even more serious problem for software support and maintenance.

SUMMARY OF PROBLEMS WITH THE LACK OF COMMONALTY AND USE OF ASSEMBLY LANGUAGE

- ORIGINAL DEVELOPMENT COST AND SUPPORT AND MAINTENANCE COST HIGH
- EXCESSIVE COST FOR DEVELOPING TRANSLATORS AND SUPPORT TOOLS FOR EACH OF THE LANGUAGES
- EFFECT OF SCHEDULE SLIPPAGE, HIGHER COSTS, AND A LESS SUITABLE PRODUCT



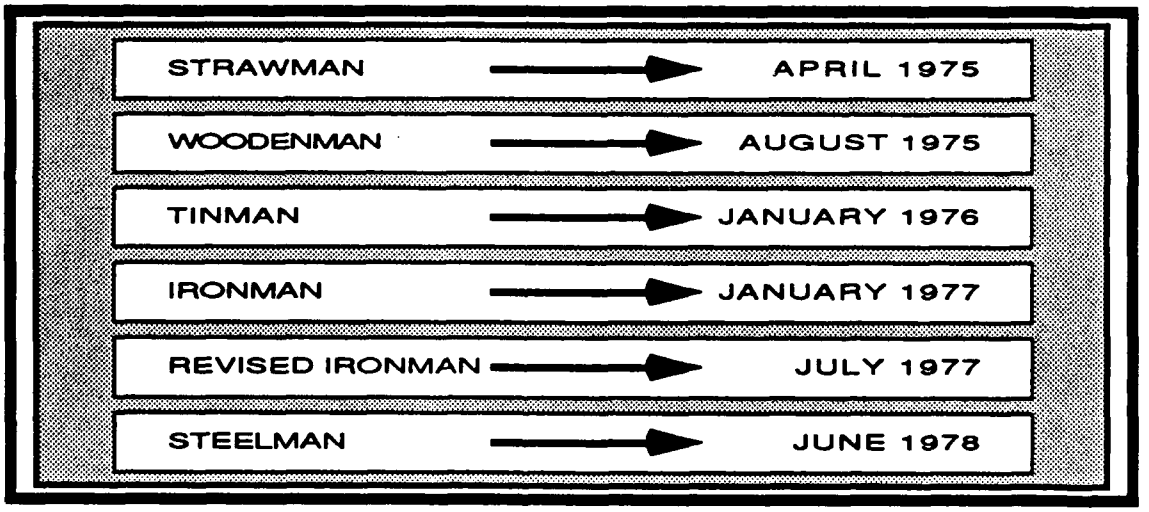
- DIFFICULTIES, TIME LAGS, AND EXCESSIVE COST TO MAINTAIN THE SOFTWARE BY OTHER THAN THE ORIGINAL DEVELOPER
- TIME LAG AND COSTS ASSOCIATED WITH RETRAINING OF PROGRAMMERS

(42:10-11)

The Search for a Common Language

It became evident that something had to be done to curb rising costs and associated problems of language proliferation. Therefore, the services began to study the feasibility of a common language. The DoD established the High-Order Language Working Group (HOLWG) in 1975 to provide a common framework in which to work this issue. The first job was to define capabilities and set forth requirements to provide the basis for a new language. The HOLWG developed a series of requirements documents:

REQUIREMENTS EVOLUTION



These documents continued to refine the new language requirements. (42:11)

The document that contained the final set of requirements was entitled *Department of Defense Requirements for High Order Computer Programming Languages: "Steelman."* The refined language requirements in the "Steelman" document were then evaluated against a number of existing languages to see if any were appropriate. In 1976 the following languages were evaluated against the "Steelman:"

- Jovial, SPL/I, Tacpol, CMS-2

  These languages were in DoD use for existing embedded computer systems.

- CORAL-66, LIS, Pearl, RTL-2, HAL/S

  These languages were used in process control systems.

- Euclid, Moral, ECL, Simula-67

  These languages were being used for research.

- COBOL, FORTRAN, Pascal, Algol, PL/I

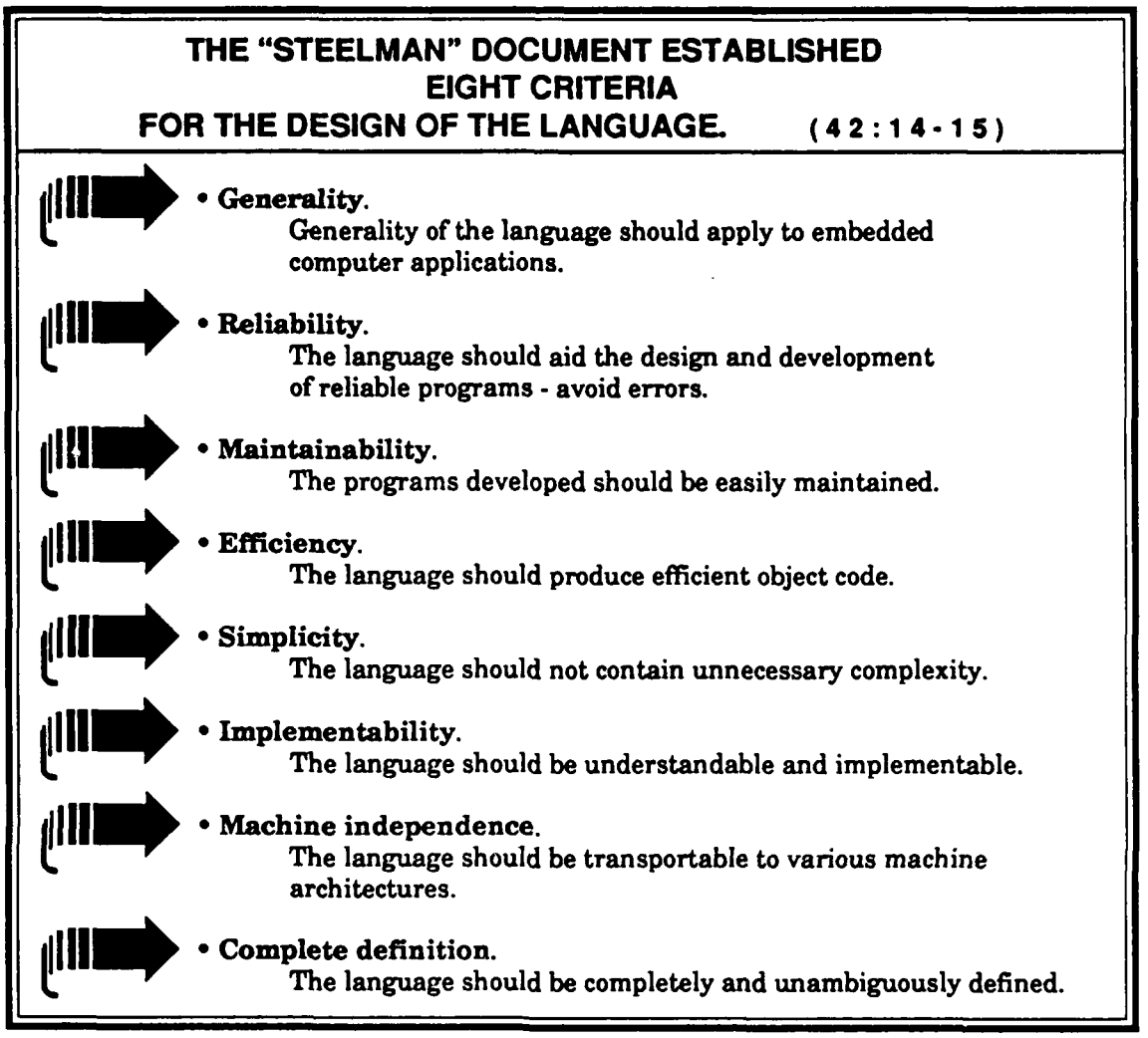
  These general languages were used in a variety of functions.

After the evaluation was completed, it was determined that none of these existing languages would fulfill the technical requirements of the "Steelman" as listed below. The evaluation did conclude, however, that Pascal, PL/I, or Algol 68 should form the foundation for the new language. (42:12)

FUNDAMENTAL TECHNICAL REQUIREMENTS
ESTABLISHED IN THE ORIGINAL "STEELMAN" (42:14-15)

- BE SUITABLE FOR EMBEDDED COMPUTER APPLICATIONS
- BE APPROPRIATE FOR SOFTWARE FOR LONG-LIVED SYSTEMS
- BE SUITABLE AS A COMMON LANGUAGE
- NOT IMPOSE EXECUTION COSTS DUE TO UNNEEDED GENERALITY
- PROVIDE A BASE FOR DEVELOPMENT, MAINTENANCE, AND SUPPORT
- EXEMPLIFY GOOD LANGUAGE DESIGN

From these technical macro-requirements came the eight criteria for the design of the language.



Creation of Ada

From the beginning of the requirements definition, layed down by the "Steelman," the DoD released a request for proposal in April 1977. Two years later the language developed by Cii-Honeywell Bull was ready for review and intense scrutiny and was coined Ada. In June 1979 the Preliminary Ada Refe: ence Manual was distributed to over 10,000 individuals for evaluation. The language went through a vigorous evaluation of test program development and recommendations for improvement. Over 100 different organizations participated in this evaluation which lead to over 900 issues raised

about the language. After a year of fine tuning, the language Ada was formally "born" on September 4 and 5, 1980. Ada then became a military standard, MIL-STD-1815, on 10 December 1980, and, in February 1983, it became an American National Standards Institute standard (ANSI). The DoD common language was named Ada in honor of the world's first recognized programmer, Countess Augusta Ada Lovelace. (42:14-15, A:vii)

Countess Augusta Ada Lovelace

Ada has been trademarked since 1981. The DoD trademarked Ada to prevent compilers that do not conform to the language standard from being sold as true Ada compilers. To insure full compliance with MIL-STD-1815, compilers using the Ada trademark must be certified through a formal validation process that is managed by the Ada Joint Program Office (AJPO). Certificates are issued by the AJPO following satisfactory completion of testing (over 2500 tests must be passed for each validation). This testing is done at two Ada Validation Facilities in the United States (one at Wright-

Patterson Air Force Base, Ohio, and the second at the General Services Administration in Washington, D.C.) or at facilities in France, Germany, or the United Kingdom. (25:735, 40:726)

Description of the Ada language

Ada is a rich, complicated programming language created to address the complex problems for which it was designed. There are no authorized subsets of the Ada language. The DoD has decided that no single subset would have the features required by all users. Ada was designed as an integrated, unified language, and therefore it is difficult to remove features without disturbing the unity of the remainder of the language. (42:viii)

Ada is a large and complex language. One element of Ada's complexity is its use of advanced and not widely-known features such as packages, tasks, generics, exceptions, private types, and others.

KEY TECHNICAL FEATURES OF ADA (40:730)

• packages	• real-time processing
• strong data typing	• exceptions
• generics	• overloading
• tasking	• separate compilation
• numeric processing	• representation clauses

Another element is its use of features common and well-accepted in the Algol 60 and Pascal class of languages but unknown to engineers with largely FORTRAN backgrounds who traditionally have designed embedded computer systems with features such as enumeration types, records, pointers, strong typing, and others. (42:viii)

KEY CHARACTERISTICS

SOFTWARE ENGINEERING PRINCIPLES (40:730)

- structured programming
- top-down development
- strong data typing
- abstraction (of data and actions)
- information hiding and encapsulation
- separation of specification from implementation
- reusability
- separation of logical from physical concerns
- portability
- modularity
- readability
- verifiability

Ada uses Pascal as a base language. Pascal was designed to be a language for the teaching of programming. This does not mean that Pascal is a subset of Ada; hardly anything from Pascal has found its way unaltered into Ada. The principal inheritance from Pascal is its basic philosophy - that both algorithms and data structures should be specifiable, precise and clear, and that the logical consistency of a program should be ensured by the compiler wherever possible. Thus, Ada is principally concerned with readability, maintainability and security. (58:11, 42:viii)

Ada, however, goes beyond Pascal in defining new types of data objects and provides additional measures to help ensure safe programming practices. Language features of Ada allow more programmer errors to be caught at

compile time rather than during execution. Ada provides capabilities for concurrent programming, error detection and handling, and effective packaging of data and procedures. It also provides capabilities and tools for large programming teams to work together effectively on large projects with a high degree of productivity. These capabilities allow Ada to meet its primary objective of being a language for embedded computer systems. (42:4)

To better visualize Ada as a programming language, the following examples are provided:

EXAMPLE OF A SIMPLE ADA PROGRAM

```
with TEXT_IO;use TEXT_IO;
procedure THIRD_MULTIPLY is
      I,J,PRODUCT : INTEGER ;
      package INT_IO is new INTEGER_IO(INTEGER) ;
      use TEXT_IO, INT_IO ;
begin
      GET(I) ;     - - read the first number
      GET(J) ;     - - read the second number
      PRODUCT : = I * J ;      - - multiply them
      PUT(PRODUCT) ;  - - print the result
end THIRD_MULTIPLY
```

THIS PROGRAM MULTIPLIES TWO NUMBERS

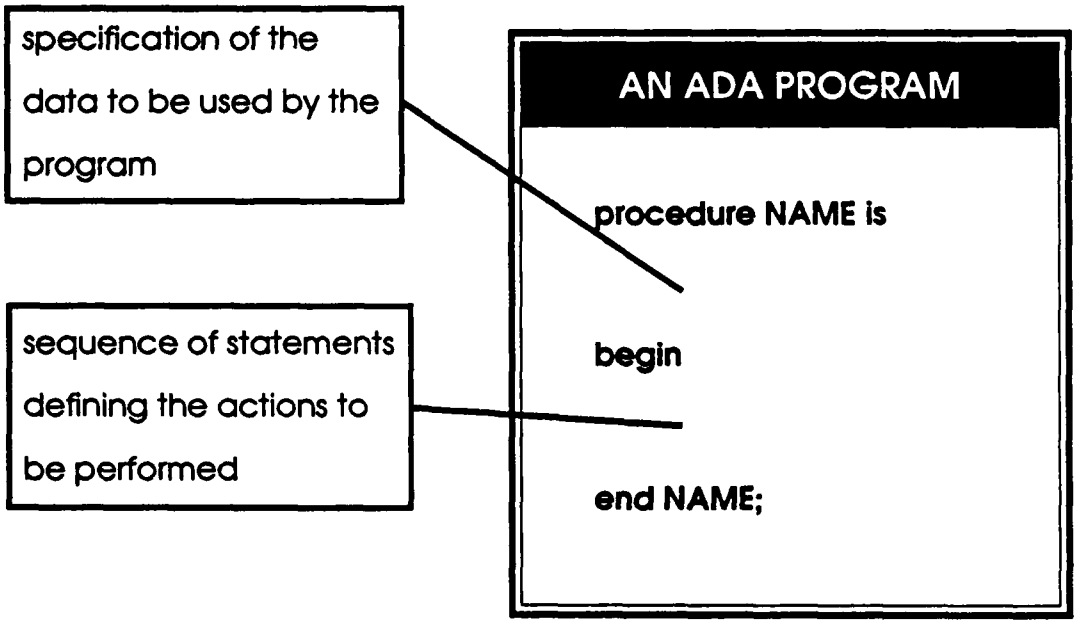
(37:3)

Construction of an Ada Program (58:14)

An Ada program requires two distinct pieces of information.

1. The data which is to be processed must be precisely defined.

2. The operations which are to be performed on that data must be specified.



procedure is a keyword followed by the name of the program which is chosen by the programmer. Keywords denote particular kinds of constructs. In this example, the keyword **procedure** denotes the start of a piece of program which is to be executed. Keywords are written in lower-case and programmer-defined words are written in UPPER-CASE.

is is a keyword followed by the specification of the data to be used by the program.

The actions to be performed by the program on the data are then written as a sequence of statements between **begin** and **end**.

The name of the program is given again at the end.

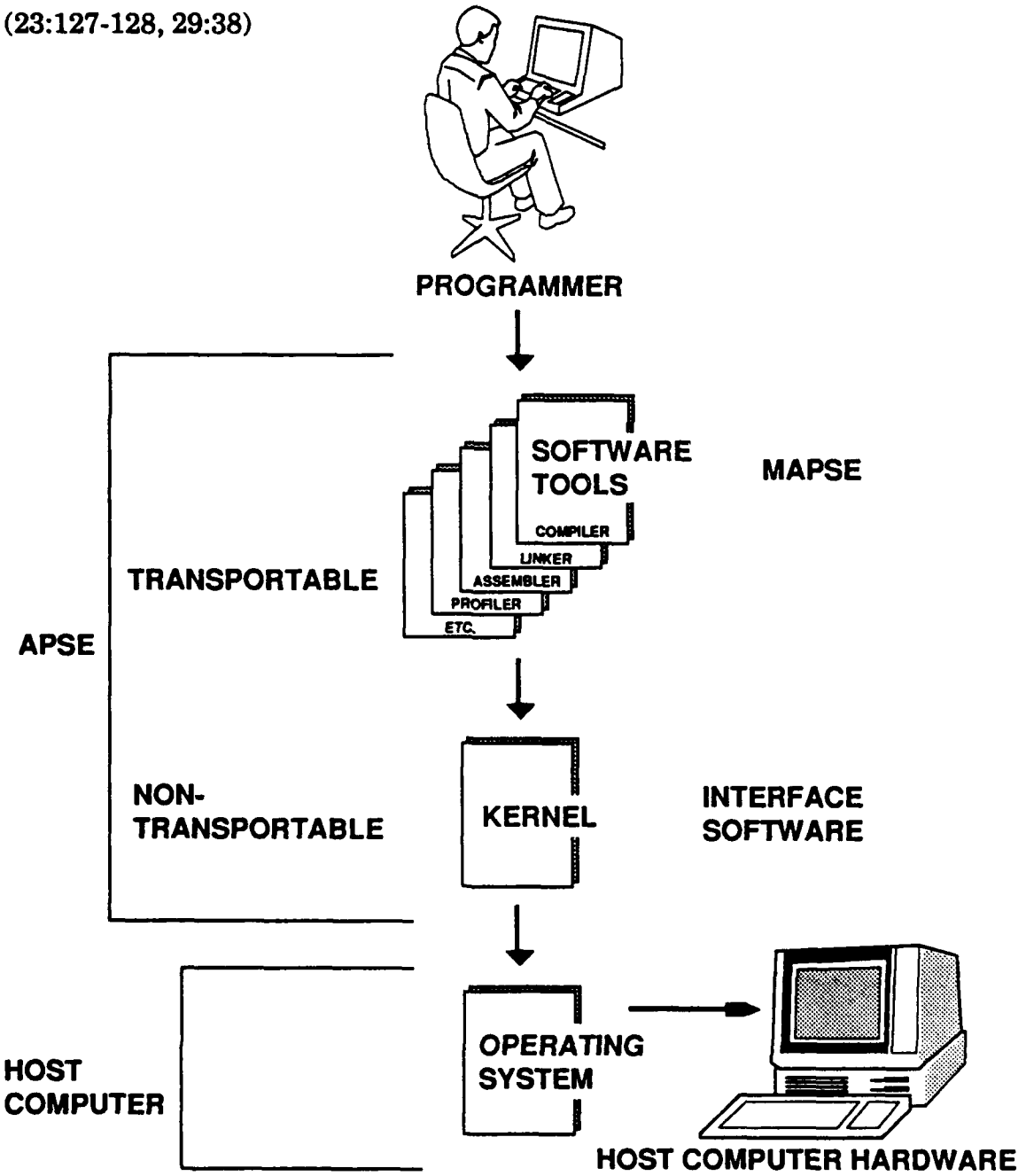
Systems designed and written in Ada use a programming approach substantially different from traditional programming. To illustrate, lets look at a problem that involves ascertaining the value of the change in ones pocket. The traditional program might say: Take a coin from the pocket; if it is a quarter, add 25 cents to the total; if it a dime add 10 cents, etc.; then take another coin. An Ada solution might say: Take all coins from the pocket and divide into like groups of quarters, dimes, etc.; determine the value of each group simultaneously; and add the subtotals together. The scope and approach of the difference between Ada and traditional programming language is substantial. (27:153)

Ada: More Than Just a Language

Designing and developing weapon system software using Ada is far more encompassing then just programming in the Ada language. To appreciate the magnitude of Ada, one must also look at its total programming environment. Since computers manufactured by different companies usually contain different architectures and operating systems, the Ada programming support environment is a compilation of a variety of software packages referred to as *tools*.

First the programmer writes the programs using a set of integrated software tools called the *Minimum Ada Programming Support Environment* (MAPSE). The MAPSE consists of compiler (with library manager), symbolic debugger, editor, job control language interrupter, link loader, and configuration management system. The MAPSE is a set of transportable software tools that can operate on a variety of different host computers. Between the MAPSE and the host computer there is the *Kernel*. The Kernel is a software

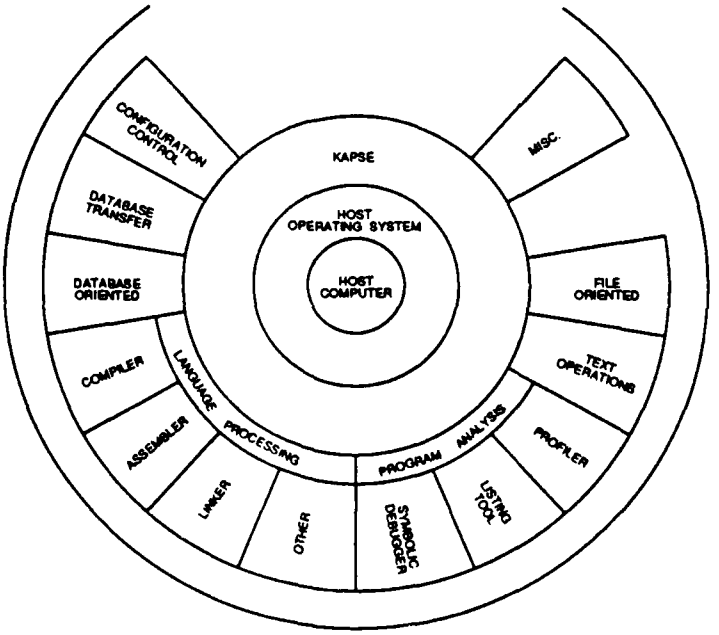
package that interfaces all of the support tools in the MAPSE to a specific host computer and is not transportable. It is unique to that computer. The software that the programmer has written flows through the MAPSE through the Kernel and finally to the host computer's operating system. The host computer's operations system then allocates the software to the host computer hardware for execution. The following diagram depicts this scenario: (23:127-128, 29:38)



Since the MAPSE is just a set of minimum Ada support tools, there is the requirement, especially in the development of large systems, to develop and use a robust software engineering approach. This approach is the interlacing of a wide variety of software development and support tools called the *Ada Programming Support Environment* (APSE). The APSE, as now conceived, is hardware independent. It is a complete, operationally integrated Ada programming environment that will support software systems throughout a life cycle which encompasses conception, design, coding, testing operational use, and modification as user requirements change. Support software entities are called *software* tools, *support* tools, or simply *tools*. (23:126)

The next diagram illustrates additional complexities of the APSE. The APSE can be visualized using concentric circles with the innermost circle representing the host computer hardware. The software tools listed on the diagram are but a representation of the some seventy different tools that comprise the APSE. (23:129)

APSE ARCHITECTURE



Clearly the development of an Ada program for weapon systems is a very large undertaking. One may question the use of such an all-encompassing, complex system of tools with the Ada language. However, as programs get larger, the system to develop those programs must be large enough and capable enough to maintain strict control over the entire devlopment process. To support large software programs throughout their life cycle, there are three basic requirements: (23:131)

- Complete and accurate information on each stage of the project from development to support.
- Both general and specific support tools.
- Configuration management.

Ada addresses these requirements through the following:

- The data base provides a repository for accurate records.
- Software tools are a part of the APSE.
- APSE provides on-line configuration control.

The importance of the Ada Programming Support Environment can be summarized as follows:

- Reduces the time to develop, code, test, document, and maintain computer programs.
- Improves program and programmer transportability.
- Simplifies project coordination and assists management.

All of the above features directly impact the cost to develop the computer program. The Ada compiler by itself is just another language, however, APSE is an integrated programming environment which is intended to reduce the overall costs of computer program development. (23:132)

The major technical characteristic of Ada is that it ameliorates most modern software engineering principles. Its unique characteristic is that it is

designed around the concept of the software component. Strengthening this characteristic is that Ada supports the production of very large software systems. In summary it is the combination of specific language features, some auxiliary technical aspects, the process that develops and supports it, and its acceptance by an international computing community of managers, users, researchers, and governments that together make Ada unique. (40:722,729)

The Ada program's administrative responsibility is now divided over three distinct entities; the triservice Ada Joint Program Office (controls the military standard), the Software Technology for Adaptable Reliable Systems program (develops techniques and tools to boost software productivity), and the Software Engineering Institute (accelerates transfer of technology to the private sector). Recently, all three entities were placed under the control of the Defense Advanced Research Projects Agency (DARPA). (39:11)

Ada is truly complex. The reader must understand that Ada is not just a language, rather it is an entire macrocosm of technology. This macrocosm of technology includes such items as the Ada language itself, the MAPSE, the APSE, and all of the controls and administrative hierarchy established to foster the use of Ada. These items, and others, make up the "Ada Culture." However, this culture is not pure; Ada has been tarnished and has become controversial . The next chapter examines the nature and scope of this controversy.

CHAPTER FOUR

THE ADA CONTROVERSY

Before discussing the Ada controversy, it may be helpful to the reader to quickly summarize the reasons for the Ada program. Previously with the proliferation of languages and the extensive use of assembly language, the following problems occurred: (42:10-11)

- High development and support costs.
- Excessive cost for developing tools.
- Unproven software tools.
- Additional costs to maintain the software by other than the original developer.
- Additional costs and time for retraining of programmers.

Requirements layed down by the "Steelman" formulated the foundation for creating Ada and solving the software problem. (42:14-15) These were:

- Be suitable for embedded computer applications.
- Be appropriate for software in long-lived systems.
- Be suitable as a common language.
- Not impose execution costs due to unneeded generality.
- Provide a base for development and support environment.
- Exemplify good language design.

The open literature clearly demonstrated to the author that there are conflicting views as to how the Ada program is doing in respect to the above.

This chapter examines the Ada program from two viewpoints. The first part of the chapter discusses the perceived benefits of the Ada program.

The second part summarizes those issues that have caused negative views of the Ada program.

Perceived Benefits of Ada

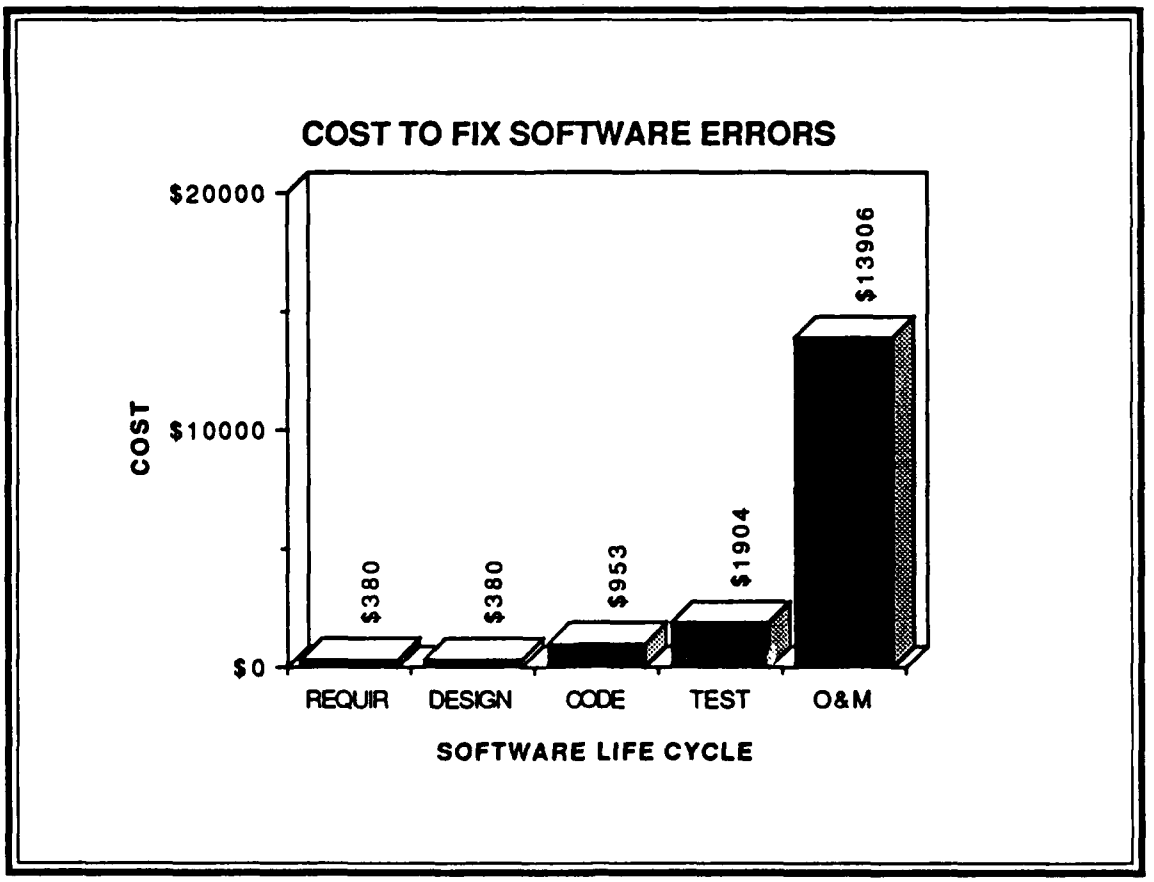
This section deals with originally expected benefits, as well as with some other benefits not entirely foreseen in the early design of the Ada effort, and gives a report on the positive Ada accomplishments achieved thus far.

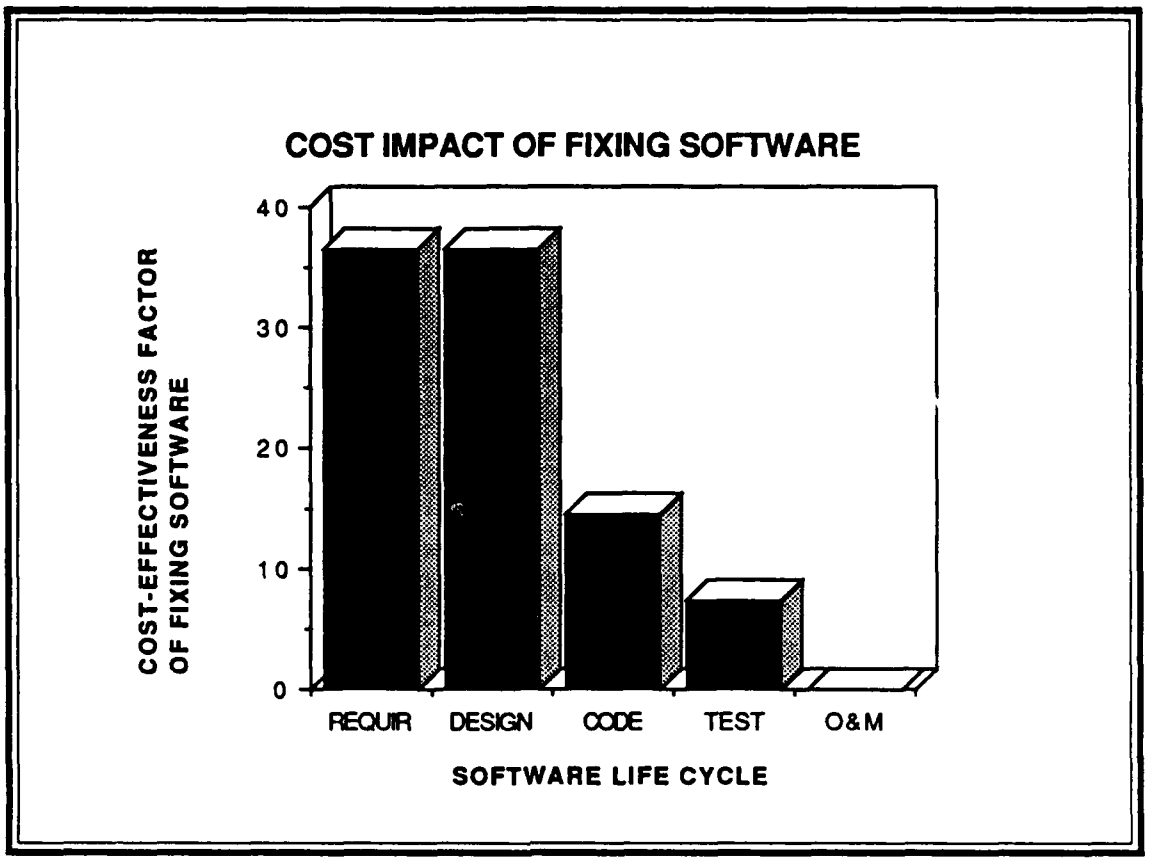
Ada's focus emerged on program reliability, program maintainability, ease of use, and efficiency. In this context, weapon system reliability is critical, since embedded computer systems typically deal with life-and-death situations. Reliability is critical because software in systems such as a flight control system for the F-16 must not fail. Ada has special features to deal with reliability. Program maintainability is important since embedded computer systems typically have long lifetimes (typically 15-25 years) and are frequently updated. Maintainability is stressed by the ease of reading a program over the ease of writing one. Efficiency is important, since in many respects Ada is competing with assembly language, and efficient use of space and time is critical in most real-time applications. (42:5)

Ada is designed to simplify the production and support of large programs. The complexity of a computer program increases rapidly as the program becomes larger. Ada provides a number of facilities for dividing programs into smaller modules thus helping make Ada programs easier to read, to write, and to modify. The ability to divide programs makes the smaller divisions entities in themselves (these entities are called software modules). As Ada environments mature, and the use of Ada becomes more widespread, software modules, in the form of Ada packages, can be developed once and

used in several systems, thus substantially improving productivity. There are however non-technical hurdles, such as incentives to contractors for reusing software and legal issues relative to reused software. Ada was designed to support reused software and early efforts are promising. (11:165, 37:vii)

Since Ada has been in existence for only a short time, the extensive use of Ada is still quite limited in the markets today. However, the reported productivity figures on a number of programs are impressive. Various studies show production rates of between 311 and 1,400 lines of code per programmer-month on a wide variety of software efforts. These figures exceed the accepted industry norm of between 325 and 400 lines of code per programmer-month. Coding and unit testing in Ada increased in productivity, especially as the number of errors generated by the Ada code went down dramatically.





Software maintenance figures are limited because of the lack of historical data. However, estimates indicate that the long term result of maintaining reused software will yield an estimated 60 percent to 80 percent reduction in maintenance problems. (11:165-171)

Although limited in marketplace *use* Ada is being *accepted* throughout the world, even in its infant stages. Ada is an ANSI and a military standard and became a U.S. Federal Information Processing Standard (FIPS) in October of 1985. Ada has also been adopted by NATO as a standard for its command, control, and information systems. Ada has met with widespread acceptance within the individual European nations as well. The policy of the Ministry of Defense in the United Kingdom currently allows either CORAL 66 or Ada and, since July 1987, new systems will require Ada. For German defense applications, only Ada and PEARL are allowed currently. The Cana-

dian government has chosen Ada as the preferred language for defense systems. Ada is also mandatory for all Swedish real-time defense systems. The real test as to Ada's acceptance will be its use in many of the new, high-profile weapon system programs, including the Advanced Tactical Fighter (ATF) and the Advanced Tactical Aircraft (ATA), along with other government programs. (25:736-738)

Since 1985 the industry trend has indicated clearly Ada's new widespread acceptance. This acceptance was also demonstrated by the wide variety of products offered at the Association for Computing Machinery's Sigada meetings. The size of the Ada market in 1984 was estimated to be about $300 million. By the end of 1985, the estimates more than doubled to more than $750 million. Indications are that the Ada market surpassed $1 billion during 1986 and projections indicate that thev will surpass the $10 billion annual rate by 1990 for DoD embedded software alone. (6:51, 56:20)

The DoD has not waivered from original objectives and continues to mandate Ada. On 31 March 1987, DODD 3405.2 established a weapon system policy which stated that all software developments for new weapon systems be performed in Ada as the single, common, high-order (level) programming language. Deputy Defense Secretary William Taft IV broadened that directive in April, 1987 in DODD 3405.1 by specifying that Ada must be used on all DoD computer resources, with a few other languages permitted in certain cases. These directives reaffirm the DoD's commitment to Ada. (21:61)

Negative Views of Ada Competence

There are, however, negative views on the competence of Ada. The

rosy picture of Ada solving the weapon system's computer software problem has been tarnished. To better understand the issues involved, we need to discuss four separate aspects related to perceptions of applied Ada. The first aspect explores various management issues and perceptions. The second discusses the problem of Ada software tool development. The third treats the highly emotional aspect of cost. The final aspect reviews those technical problems that surfaced with the application of Ada.

Management

Ada has become an emotional issue even at the senior levels of the DoD. The following quotes from the incumbent Under Secretary of the Army, James R. Ambrose, at the annual Ada Expo and Special Interest Group convention in Boston sums up many feeling about Ada. He cautioned the Ada vendors to be conservative in statements about the capabilities of the new language because he felt that exaggerated claims do more harm than good. (21:60) Typical of his comments are:

> Ada is beset with too much Hyperbole for its own good.
>
> I am not impressed, have not been, and I remain willing to be impressed, with quantitative measurements of the superiority or productivity or the virtues of anything such as Ada.
>
> What I have seen thus far has been largely rhetoric.

Using this new language met with strong opposition from many program managers, especially in the early 1980s, when Ada compilers were either nonexistent or of poor quality. This caused program managers, both in the military and in industry, to write Ada off as a failure. In addition to

these early assessments, many hard line program managers believed that a well-structured, highly maintainable code could be produced in any language, as long as the programmer worked within the constraints of an effective design methodology. (49:15, 21:60)

Critics contend that the method that brought Ada into existence was lauded as unique - this is true by any quantitative measure - but qualitatively Ada was not so different than every language before it. Ada was an exercise in bottom-up logic. With all of the work in the requirements document, there was no clear expression of how Ada was supposed to be used. If Ada's gestation really was unique, why was there a long period of intensive investigation of how to use Ada effectively following the completion of the language? This was especially true in the need for a proper software development environment to achieve the full benefits of the language. The requirements analysis for the programming environment was not begun until the language definition was essentially complete. This has been akined to placing the "cart before the horse." The discussions continued throughout the early 1980s point to the assertion that, after a decade of effort involving 27 different contractors and a reported $1 billion in taxpayer money, Ada has become an electronic tower of babble. The main reason given for this assertion is that the DoD never provided complete performance specifications to the contractors. (20:87, 14:280)

Tools

Ada has been overpromised based on the time it takes to mature the language and the new engineering practices. A variety of software tools need to be developed in order to insure the full benefits of Ada. A large amount of

these tools are still not fully developed, especially existing support tools which provide the following capability: (33:20)

- software documentation.
- configuration control.
- maintaining developmental history.
- debugging.
- project schedule and effort management.

Costs

Acquiring an Ada capability represents a large corporate investment. The initial buy-in, however, is only the beginning of the financial burden. Corporations must also consider how long the training will take, and how productive the programmers will eventually become. These costs not only include the cost to train programmers in the Ada language, but also to train the programmers and mangers in new software engineering practices and disciplines. Until Ada compilers mature, the slower compilation time and run times of Ada add to development costs. After considering training and productivity issues, it soon becomes clear that a bargain-basement Ada system may be a more expensive proposition then buying a robust system in the first place. (33:17, 24:101)

A company just starting to use Ada is very concerned with training. Ada cannot be taught the way older, sequential languages were taught. Considering Ada as just another programming language is like viewing the tip of an iceberg. The education of Ada professionals will take time. A commonly accepted training time frame is from six to 18 months in order to achieve proficiency. This is because Ada is one of the most complicated programming

language in existence. Since Ada must be taught in a software engineering context, and not just as a language, educating the existing work force in software engineering will not be easy. (27:148, 153)

Procuring an Ada capability is a major investment that requires a thorough examination of six basic issues: (24:101)

- compiler validation.
- implementation dependent features.
- bench-marks and performance.
- development environments.
- retargetability.
- training.

Because of the many uncertain and unproven techniques associated with Ada, the cost to do business using Ada is suspect at best.

Technology

Embedded computers in modern weapon systems and its ensuing avionic software demand very fast processing, swift calculation of floating-point mathematics, and compilers that produce compact code. Most validated Ada compilers produce code that is too inefficient for avionics. Compounding this is that many avionic systems contain microprocessors or custom arrays and have no Ada compilers targeted for the smaller computer. (48:45)

Many of the first Ada compilers were written by software engineers with experience in large-system design. The use of features specific to particular implementations of the language in avionics and a lack of experience and information exchange among compiler writers has started the application of Ada down a rocky road. Compiler writers are not used to considering the

underlying hardware. (48:45)

The three examples below demonstrate that the Ada compilers written for early embedded systems fall short of actual operational requirements.

• A missing capability of the Ada compiler is interrupt handling. Ada treats an interrupt handling routine like any other task. An Ada compiler will generate code to allocate data structures, save and store data from other tasks, and so forth, when responding to system-generated interrupts. In real-time control applications, such overhead can slow down the processing, rendering the control useless. There is a compiler implementation alternative that can be tasked to do something out of the ordinary at a specific point. This in turn does not generate the housekeeping code for the interrupt. This technique however, is not tested during the compiler validation test. (48:46)

• Ada in embedded systems needs to site data or code at an absolute memory location. This is a primary feature for embedded systems. This technique is described in the requirements documents but not validated on early compilers. (48:46)

• Ada lacks the explicit bit-set and bit-reset operations to manipulate external hardware devices in microprocessor systems. Bit manipulation - the direct issuance of a command to set an individual bit in memory - is crucial for quick response in systems with bit-mapped I/O. This is an optional feature in the requirements documents that is not included in many compilers. (48:46)

General Dynamics engineers have learned in developing a program in Ada for a series of flights on an F-16 testbed aircraft that the language has some limitations in running real time programs. As the General Dynamics engineers have found, Ada today faces two distinct constraints. (15:73-75)

• The first constraint is that most of today's embedded processors are

relatively compact microprocessors having a very limited memory and space requirements. Secondly the operational programs run at very high speeds and must be able to react to the continuing changes in the physics of flight. Ada is a very robust language and is not as yet optimized for smaller processors, which makes programming of small processors in Ada difficult. In addition, "hard" real-time applications that require analog to digital conversions in less than 10 microseconds require a language that is optimized in the run time environment. The General Dynamic's tests found that Ada has its limitations in these "hard" real-time applications.

In summary, the literature reveals that the Ada system continues to expand and is beginning to be accepted in the community, however, there are a number of issues and problems that critics raise to question the competence of Ada.

CHAPTER FIVE

CAUSES OF THE ADA CONTROVERSY

This chapter will address the underlying causes of the Ada controversy. First the conceptual rational which has fostered the controversy will be outlined. Second, various issues will be assessed and their validity will be determined. Management, tools, costs, and technology will be the bases for the discussion.

Why There is a Perception That Things Are Not Going Smoothly

Ada was designed to replace existing languages for the programming of DoD embedded computer systems such as Jovial (USAF), CMS-2 (USN), Tacpol (USA) and SPL/I (USN). In addition, Ada is also intended to replace FORTRAN where it is used in embedded computer systems applications. Ada makes the entire software business different then it has been for the past thirty years. One can expect that this dramatic change will foster animosity across the government and industry. This subsection addresses, in the broadest terms, the causes for the controversy. (42:4-5)

Management

Fundamental changes often meet with great resistance. Since Ada impacts attitudes, mindsets and cultures, its implementation can be expected to generate substantial resistance. Counters to the arguments have been encountered at every level in government and industry. Fear that program-

ming skills will be devalued and fear of downgrading, coupled with the refusal to believe that the new methods actually work, have to be dealt with. This fear of downgrading may be countered by the fact that software engineering is actually an upgrading of skills and that is is easy to maintain the software product written in Ada code. Therefore, one has to expect resistance. (27:153)

The essence of software is a construction of interlocking concepts: data sets, relationships among data items, algorithm, and invocations of functions. This essence is abstract, in that the conceptual construction of a software program is the same under many different representations. It is nonetheless highly precise and richly detailed. The hard part of building software is the specification, design, and testing of this conceptual construction itself, not the labor representing it, and testing the fidelity of the representation. We still make syntax errors, to be sure; but they are fuzz compared to the conceptual errors in most systems. Building software will always be hard. There is inherently no magic in software programming. (33:51)

The development of the Ada market and its acceptance emanate from four major forces: government policy, technology, market dynamics, and industry economics. These forces have shaped the Ada environment of today and will continue to shape it into the 1990's. (35:41)

The evolution of Ada comprises four stages:

- Proof of the technology

  1972-1983. Tremendous uncertainties existed

- Early stages of commercialization

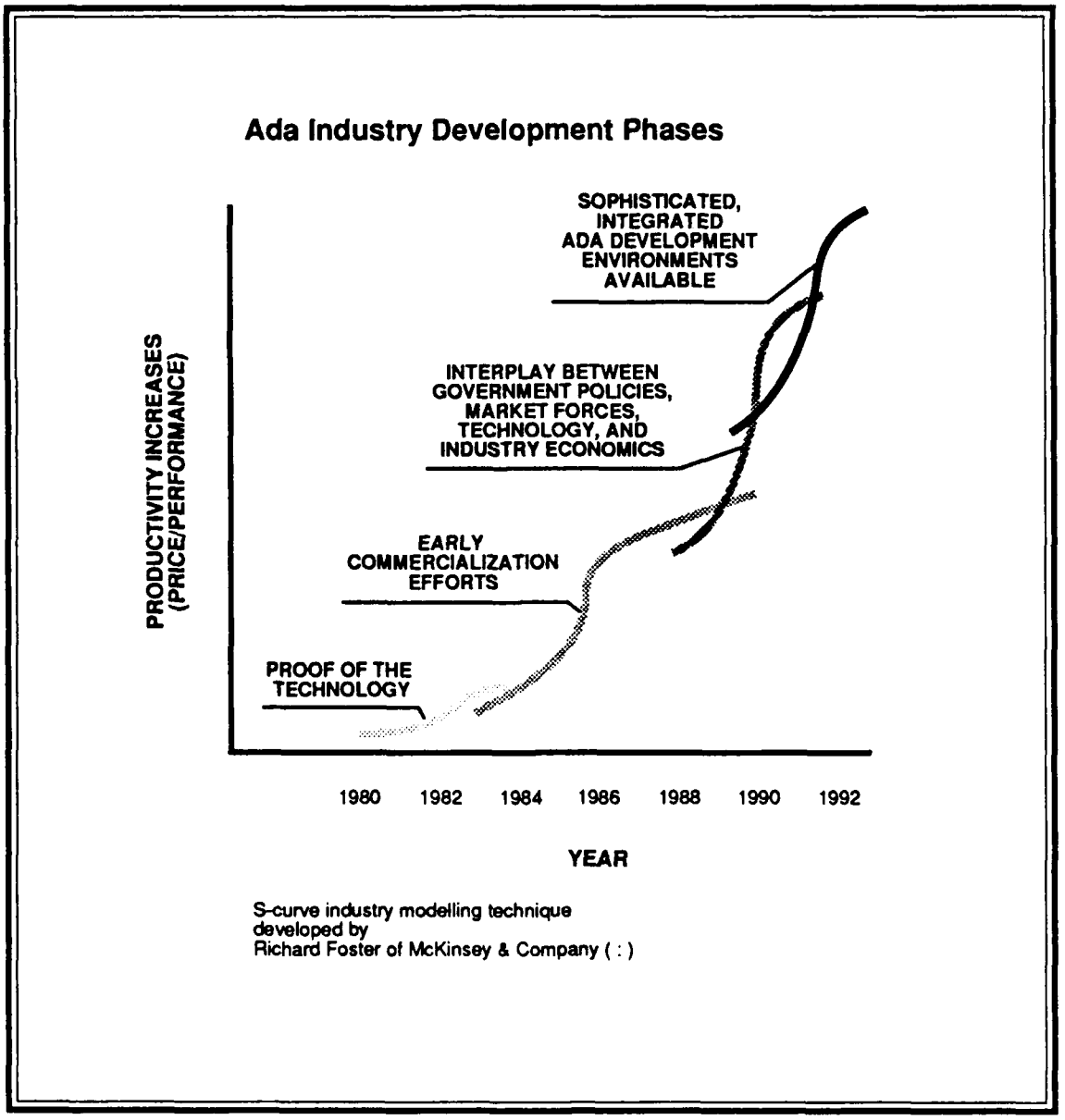
  1983-1988. Government policies were strongly worded but waivers were granted. Companies committed to the Ada industry began to emerge.

• Transition to industry maturity

1988-1992. Strong government mandate. Prices are dropping. Commercial adoption is increasing. Universities are teaching. Senior management is beginning to understand the economics.

• Commercial APSE

1990's. A sophisticated, integrated Ada development environments should be available for a variety of hardware platforms. (35:41-43)



Structuring a software development program based on Ada is quite different then using older methods. The development process requires close attention to software engineering principles to recognize and control the complexities of the Ada programs. The disciplines that must be included in an Ada development effort include: accurate and integrated knowledge of the system, program and system objectives, and the design, development, test, maintenance and modification process, including the management process. (38:32)

In September 1987, the Defense Science Board published a report on the finding of their Task Force directed to study military software. The key points made in the report are summarized:

- The Task Force was convinced that today's major problem with military software development are not technical problems, but management problems. (33:7)

- The Task Force reported that it is very important for DoD to have a standard programming language; Ada is by far the strongest candidate in sight. The 1983 mandate for Ada was technically premature. DoD commitment to Ada since that time has been weak. The state of Ada compiling technology is now such that it is time to commit vigorously and wholeheartedly. (33:4)

- The Task Force recognized that few program managers will want to take on the headaches of being the first user of a new tool. Only top-level DoD commitment and mandate can make this happen. (33:2)

Despite industries like IBM embellishing Ada, there is still animosity in the community, and Ada has neither extended its identity nor ventured beyond the military/industrial world. One reason is the entrenchment of languages such as COBOL in data processing departments. Another, is the

relatively recent incursion of so-called fourth-generation languages from a number of vendors. Says Michael Ryer, director of Ada products for Intermetrics Inc., Cambridge, Mass., "There is a great inventory of billions of lines of COBOL code. You can't very well rewrite every single line, and it's hard to put a little bit of Ada on top of a lot of COBOL." (32:35)

Tools

The problems with software tools (availability, reliability, robustness, maturity) will remain a controversial issue for years to come. The development of the tool set takes time, not only from the production of these tools, but also from the appropriateness of the tools. On one hand, the advocates of Ada point to the availability of tool sets, and on the other hand the critics of Ada point to the lack of such tools. It appears that they are both wrong and both right.

Costs

The cost associated with training managers, engineers, and programmers varies somewhat from company to company. Organizations that were knowledgeable in a language such as Pascal find that Ada is merely the next logical step in language evolution. For those with backgrounds in FORTRAN, COBOL, or assembly language, Ada, for many, is too much to swallow in one bite. (42:x)

The Ada language was designed to be one of the primary means of increasing software productivity and controlling costs. Because of the intense

interest in Ada, programming efforts have been carefully monitored to gauge the extent of progress in increasing productivity. The results of recent studies reveal results that clearly demonstrate Ada's link to high productivity. However, increased effort in the design phase has been necessary to produce software for potential reuse and the Ada features that support modularity. Therefore, many of these software production efforts have indicated a shift in effort from the classical coding phase to earlier in the life cycle. But developers have noted that increasing the effort in design is more than offset by decreased effort in coding, integration, and maintenance. (11:165)

Technology

Ada is a modern language system designed to meet the needs of software engineers programming applications in embedded computers for many years to come. Virtually all existing languages in general use were designed a long time ago. Even Pascal, is now almost two decades old. When these languages were designed, computer systems had restricted memory space, microprocessors were almost non-existent, and multi-processor configurations were rare. Today, however, Very High Speed Integrated Circuits (VHSIC) and Very Large Scale Integrated (VLSI) technologies offer potentially unlimited processing power and memory capacity, with the result that more and more complex applications are being undertaken. (58:11)

The reputation of Ada has been tarnished somewhat by actual tests using embedded computer systems in operational weapon systems. In September 1984 an F-15 Eagle fighter plane had, for the first time, used Ada successfully in the field. Software that was coded in Ada operated a digital

flight control system. Technical difficulties that were annotated included: (48:44)

- Inefficiencies for embedded systems; it produced code that was too large and too slow.
- It lacked some essential features: bit manipulation, interrupt handling, and the ability to store data at specified locations in memory.
- Programmers reported that some software packages could not be transferred between systems without modification.
- Programmers who could code in Ada were scarce.
- There were few good compilers, debuggers, and text editors suitable for the needs of embedded systems. (48:44)

Ada's complexity has contributed significantly to the slow maturation of the language, its compilers, and associated tools. Ada compilers execute slowly compared to compilers like FORTRAN compilers. However, Ada compilers do much more checking of the code and hence find more errors in the program. In addition to the compiler taking a relatively long time to compile, the code generated by the Ada compiler is not highly optimized. Ada offers the capability to do dynamic checking while executing, thus slowing down the execution speed. If this checking is turned off, then the code will run quite a bit faster. Eventually as the market expands and matures then we can expect that more optimized code generators will be forthcoming. (33:17)

The initial institutionalization of Ada has been wrought with opinions that question the validity of the DoD's decision to adapt Ada as a standard. The next chapter will interpret the findings of the research and address those concerns.

CHAPTER SIX

AN INTERPRETATION OF THE FINDINGS

This chapter places the controversy into perspective as it relates to the intended purpose of Ada. If one agrees that there is a software problem then something must be done. Not only from the efficiency, reliability and capability of modern day programming, but there are marketing forces outside the United States that are cause for action. In Japan, for example, the software productivity norm is 3,500 lines of code per man/month. In the United States, it is 183 lines of code. The same thing could happen to the United States software industry that happened to the United States home electronics, steel industry and to others - the U.S. lost its lead. We must not fall victim to stagnation, we must find solutions to increase productivity and sustain our lead in technology. (9:52)

Management

Throughout the research on this paper it became fairly clear that the software community centered their attention on the programming language Ada and therefore shunted the greater aspects of the Ada system - that of improved software engineering technology. Embellishing the software engineering technology is crucial to Ada's success. (6:44)

Developing software in many aspects is very abstract. Producing code and allowing that code to perform its intended function historically has been the measure of success. Therefore, the software community places a lot more emphasis on the act of coding than on any other phase of the software life

cycle. For many managers and programmers, no work is performed until code is produced and the computer does something. It is much simpler to require a programming language like Ada than it is to require a software engineering discipline which embellishes Ada. That is why many people view DODI 5000.31 as a precept to continue "business as usual" with Ada being the implementation language. Not only is the understanding skewed but the amount of effort required to educate the software community in the software engineering discipline is staggering compared to the amount of effort required to teach Ada syntax. Therefore, for many people Ada is just another language and the thinking as to its benefits falls within this limited scope. (6:44-45)

The transition from limited software engineering practices to more rigorous software engineering methodology is challenging. Government organizations and contractors cannot expect Ada to be "just another programming language" to be learned after a contract is awarded. Many of the software managers today still believe that you can manage a system development using Ada with the techniques and tools of older languages. The key to successful efficient software development in Ada requires advance commitment by the contractor to teach managers and developers the language, its software engineering features, and how best to use it throughout the software development life cycle. (17:474, 1:65)

The software engineering aspect of Ada has not been helped out by the validation of Ada compilers. Even though the validation consists of running more than 2,000 test programs to ensure that the language meets the requirements of MIL-STD-1815A, this does not mean all functions necessary to do a program are available. Chapter 12 of Appendix F of Mil-STD-1815A, specifies the machine-dependent functions are not part of the validation. The

speed and efficiency of the compiler also are not checked. Other functions and items not checked during the validation process include the quality of the environment tools, tasking, low level I/O, and library manager. This concentration on the syntax of the language and the apparent disregard for the software engineering environment just added fuel to a misunderstanding of what is required to work with Ada. However, plans are under way to develop a full-scale Ada Compiler Evaluation Capability (ACEC) that will address all of the performance aspects rather than just syntax conformance aspects of Ada compilers. (10:54, 29:38)

Sometimes it takes a "kick in the pants" for major corporate acceptance of a new way of life. International Business Machines (IBM) signed agreements with a number of Ada software and hardware vendors to make it easier for aerospace companies to use IBM computers to design and run software in Ada. This was in response to IBM's loss of major contracts such as the software support environment for the NASA space station. Valued at $140 million, this contract would have required the development of about 10 million lines of Ada code. IBM's shift to Ada acceptance has been slow but now IBM's long range plans are to provide Ada software for all of its computers. (22:67-69)

The Defense Science Board Task Force on military software stated that the DeLauer mandate to use Ada was premature. They felt that it could not be followed in 1983 because of slow maturation of the language and its compilers. However, Dr. DeLauer had not issued that mandate, it is doubtful that Ada would be as far as it is today. (33:17)

The DoD mandate to use Ada is stonger because it is getting harder and harder to get a waiver. This headlined thrust, coupled with acceptance throughout the world, will aid in the transition process. Wider acceptance is

now becoming a reality with NATO, Japanese, and civilian software houses, including the FAA's Advanced Automation Program to upgrade the air traffic control system and NASA's Software Support Environment for the space station. Ada still has a way to go in order to claim a wide field of acceptance. Ada tools, particularly compilers, are still not optimized to meet the special needs of embedded systems. The Software Engineering Institute at Carnegie-Mellon University is developing a software engineering curriculum to reduce the shortage of qualified software engineers proficient in Ada. This should also help. (43:43)

In the broadest management terms, one can look to the aerospace industry and see why the transition to Ada has been in the limelight of controversy. After examining some of the issues, the need for Ada gets stronger and stronger. (16:53-68)

- A casual attitude about software and software organizations.

Typically the hardware organizations have a great depth of knowledge of the system being designed. Software on the other hand is performed in the aerospace industry by people with specialized training in software with little training in the system to be designed. This does not work with Ada systems.

- The glamor days of the aerospace and software-economy, prestige, and draft are over.

The brightest people used to go to the aerospace industry. Today the situation has changed and the software people are being drawn to the commercial industry.

- Dealing with the abstract is difficult.

Not everyone can learn Ada. Ada is the first language intended for widespread use that emphasizes formality and abstraction. It is wrong to expect that everyone who is presently developing software can learn Ada.

• The corporate culture of cost effectiveness does not encourage quality.

Corporate entities in the United States exist for one well-defined purpose - maximize profitability. Investments in prototypes, test and development equipment, and building the software multiple times as reusable packages are rare.

• The DoD Customer-User-Vendor relationship is a hostile one.

The program offices now mandated to procure systems designed and built in Ada do not understand the motivation for Ada nor its potential benefits. The buyer is acquiring Ada-based developments in an environment fostering hostility rather than trust.

• Bureaucracy as a substitute for decisions is prevalent.

A mandate via standards as a means for encouraging Ada use will not foster the creative process and therefore will not generate enthusiasm within the program offices or the development team.

• Creativity and Quality take a back seat to short term management concerns.

Short term profits take a front seat in lieu of the eventual use of the software.

Tools

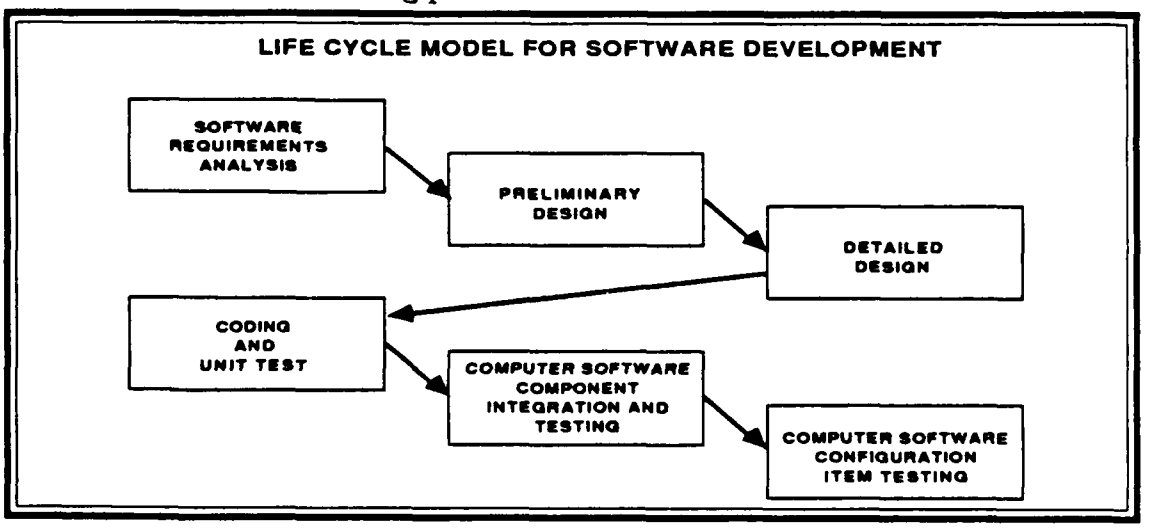
Concerns over the efficient and effective use of software tools hold the most credence in downplaying Ada. After July 1984 all Mission Critical Computer Resource programs for the DoD had to use Ada. However, at that time there were not many efficient support software tools available to help the program manager or, for that matter, the programmer comply with the

mandate. Compilers were being developed, most of which were not even targeted to the required processor. The attempts to define environments appeared to be slow and cumbersome. Yet, by mandate, DoD weapons systems had to use Ada.

The advantage of this approach was that it got people's attention and convinced the community that the DoD was serious about Ada. However, because of the lack of efficient software tools to support an Ada development effort, program managers from industry and the government had real concerns about cost overruns and schedule slippage. These problems, coupled with a lack of trained Ada programmers, led to a reluctance on the part of many program managers to heavily commit their programs to Ada. The waiver process was established to defuse the concerns for these managers. Since that early time, as more and more programs embellish Ada, the waiver process should cease and the tool environment should strengthen. (12:36)

Costs

DOD-STD-2167 contains a life-cycle model for software development which includes the following phases:



The classical cost percentages change with the implementation of Ada. Software requirements analysis for a representative Ada development effort takes about 15 percent of the total effort. This is a higher percentage than in typical projects. The design phase of a software development effort using Ada goes up compared to typical projects. However when reusable Ada designs are implemented, productivity increases of 50 percent have been realized. Therefore, program managers that are used to costing their development using older languages will have to reevaluate expenditures during the development process. The total cost may even go down as the Ada methodologies mature. (11:166-167)

We mentioned in a previous section of this paper the cost of training and emphasize that it is mistake to treat Ada as just a language; it's a methodology. A course in the language C may take three days. It takes a minimum of 21 days to learn just the syntax for Ada. This additional cost must be budgeted for and its benefits must be realized to be accepted. (49:15)

Technology

Technically speaking Ada has finally come of age. Over the past three years, the performance of Ada systems have improved dramatically. Many compilers now have performance characteristics better than older languages such as Pascal, and offer many other benefits for the real-time programmer. The literature suggests that there is a definite increase in processing capability of thc newer compilers. This is refreshing, since the technical side of the software community must embellish Ada as much as the management side. (30:76)

Recent application testing of Ada programs has verified the continuous

improvement of the language. In August 1988, a 70-hour joint Air Force and Navy test series was flown on an F-15 at Edwards AFB. The tests were part of the Ada-Based Integrated Control System (ABICS) program. (41:16)

A major objective of the ABICS program was to establish whether an Ada-based system could respond fast enough to match the performance of standard units programmed in assembly language or FORTRAN. The recent ABICS test established that, when combined with compatible computer hardware and software compilers, Ada does not impose any significant restraints on system speed. This is critical as it can now be applicable to flight-critical avionics as well as other time-sensitive avionics functions. (41:16)

Much of the previous adverse press on Ada has begun to settle down. Management is beginning to understand its importance and the technical community is beginning to use it effectively. Only time will tell to what extent the decisions made in the last decade will affect our war fighting capability for the future. The next chapter will conclude with an assessment of the original hypotheses and make recommendations for the future.

CHAPTER SEVEN

SUMMARY AND CONCLUSIONS

Summary

This is the concluding chapter of this paper, but not the end of the Ada controversy. Ada is still maturing and will have to go through many more growing pains before the controversy settles down. The author stated the purpose of this paper was to address the subject of the still new and controversial Ada, asking the question whether it will it solve the perceived software problems for tomorrow's weapon systems, or will it cause unmitigated problems?

The number of computers embedded in DoD weapon systems continues to escalate at unprecedented rates. In weapon systems of the recent past, hardware was the driving force in system capability, cost, selection, reliability, and maintainability. Currently emphasis shifted to software as the preeminent focus for new weapon systems because of software's potential to increase the capability of the weapon system and the high costs incurred during development and support phases of a weapon system's life. Reviewing the purpose and objective of the study centers on the fact that Ada is still a relatively new and controversial programming language. The problem under consideration in this study was that of determining the extent to which Ada has achieved its intended purpose of functioning as a standard programming language for the creation of large-scale embedded computer system programs for DoD weapon systems. Consequently, this study examined the development since its adoption to the present and actual performance of Ada in

actual use today as well as in developing future programs for weapon systems to determine whether this programming language has corrected the perceived software problems of past software languages or whether Ada use has repeated the problems of past computer languages as well as caused newer problems that nullify achievement of its selection objectives.

Chapter I highlighted the problem, purpose, objective and hypotheses of the study. The chapter also explored the ever increasing dependance of our modern day weapon systems on the computer and its ensuing software. With this escalation, and the dependency of computer systems, comes the high and ever increasing cost associated with developing and maintaining weapon systems. The final part of the chapter introduced the DoD solution to the software problem - Ada.

The next chapter unraveled some of the computer mystery and established a common framework to discuss the subject of Ada. The last part of that chapter directed the attention of the reader to the ever increasing importance that the computer plays in the critical functions and capabilities of our modern day weapon systems.

The third chapter set the stage for understanding just what Ada is all about. An examination of some historical aspects of Ada development and use were provided to give a framework for the analysis. The Ada engineering support environment and its ensuing software tools were reviewed to scope out the complexities of using the language.

The fourth chapter examined the Ada program from two view points. The first part of the chapter discussed the perceived benefits of the Ada program. The second part summarized those issues that have caused negative views of the Ada program. To better understand the issues, the section was broken up into four subsections discussing the negative perceptions of

applying Ada.

Chapter five discussed the underlying causes of the Ada controversy and why the controversy arose in the first place. The next chapter placed the controversy into perspective as it relates to the intended purpose of Ada.

In summary the issues were explored in a broad context in order to get a handle on the question at hand. Ada, still new and controversial, will it solve the perceived software problems for tomorrows weapon systems, or will it cause unmitigated problems?

Conclusions

It is appropriate to review the premise of the study hypotheses. The Department of Defense embarked upon a firm direction to install Ada as a cornerstone to solving the "software problem." Since the infusion of computers into U.S. weapon systems, controlling software costs, software timeliness, and reliability have been rout with less than desirable results. Solutions to past software problems have not worked. Therefore, this study had two major hypotheses:

H_1 = Ada has achieved its intended purpose of functioning as a standard programming language for the creation of large-scale embedded computer system programs for DoD weapon systems.

H_2 = With the proper degree of software engineering, Ada, used as a standard computer programing language for weapon system software development, can eventually resolve past

software language problems in the areas of cost, reliability, system capability, maintainability, and weapon selection.

The author concludes that hypotheses one is true. There is overwhelming evidence that Ada has achieved its purpose of functioning as a standard programming language. This is true based on Ada's mandate, its growing use in modern weapon systems and its acceptance in the international community.

The author concludes that hypothesis two may be true. Clearly Ada was designed technically to address the issues of solving the past language problems of reliability, system capability, maintainability and selection for embedded computers in weapon systems. However, the issue of cost reduction was not totally supported by the research. On one hand, considerable resources are required to allow the developer the capability to program effectively using Ada. On the other hand, there appears to be a future saving potential associated with the use of Ada. Both start up and support costs were not effectively substantiated during the research, therefore, the author cannot totally assess the full validity of the second hypothesis.

Recommendations

The insertion of Ada has started to take hold in the DoD, even though it was by decree. However, much more must be done in order to foster greater acceptance of this language.

The author offers the following recommendations:

- Since there is overwhelming evidence that Ada has achieved its purpose of functioning as a standard programming language. The full benefit

of using this language will be forthcoming only if the DoD and the aerospace industry foster the transition of new weapon system development into an Ada environment. This is not to say that the criticism should cease, on the contrary, criticism spears innovative reactions and this too can build a stronger system to control the software problem.

- The issue of cost was not totally supported by the research. Until this issue is empirically proved one way or the other the Ada controversy will continue. The problem with proving this issue is that time may be the only avenue for proof. Most of the research points to a great saving in the operational and maintenance phases of a system's life cycle. This is assumed because of the software engineering, ease of maintenance, and reusable software aspects of Ada. However, Ada has not been around long enough to substantiate these claims. Therefore, it is recommended that a cost development history on a number of major Ada design efforts be monitored by the Air Force and updates be published as the weapon system goes through its various phases of development and operation.

- A final recommendation looks to the future. If Ada and all of its claims come true, we must not sit idly by and be complacent to claim Ada as a victory for generations to come. Computer technology continues to escalate and change. The DoD must keep abreast of these changes and react to them. As graphical and automated languages bring us into the next generation of software, the DoD should invest resources to see what the next solution to the software problem may be.

A Final Word

The research suggested that indeed, Ada was not the calamity, but a

closer step to the panacea. Ada's progress has been viewed as slow by many. However, when one considers the impact on system design, operational capabilities, the escalating presence of computers on weapon systems, the critical nature of the missions the software must produce, and the costs associated with software development and support, it is prudent to go slowly to get the best product we can. The Ada market continues to expand. The Deer Isle Ada Research Group of Stonington, Maine, recently released a two part book entitled *Ada Market Reference: A Guide to the Industry*. This book will be updated every two months and gives a comprehensive review of the status of Ada with vendor products and services. Anyone interested should contact Lisa Schoonmaker at 207-367-5828. Ada has progressed from a program focused on the development of a high-order language to a program embellishing that language and all of the good software engineering principles that must be applied to today's weapon systems.

It is predicted that Ada will be the most important programming language of the 1980s and 1990s and the last new major language prior to automatic programming. We can not afford to wait around for these new automatic programming systems, because we have to take care of the problems at hand today. (27:148)

GLOSSARY

Analog: smoothly changing physical variables.

Assembler: a program that translates an assembly program into machine code (object code).

Bus: a set of wires that carry signals around a computer or digital electronic system.

Compiler: a program that translates a high-level program into machine code (either directly or through an assembly program).

Digital: pertaining to the representation or transmission of data by discrete signals.

Digital computer: a machine that operates on data or instructions expressed in discrete, or on-off (one or zero), form rather than the continuous representation used in an analog computer.

Exceptions: a method which identifies an error in a program while it is executing.

KAPSE: unique to a specific computer, the Kernel Ada Programming Support Environment provides the minimum set of functions necessary to interface the host computer to the rest of the APSE.

Large-scale integration: the placement of thousands of logic gates on a single integrated circuit (chip).

Microprocessor: a single integrated circuit containing all the elements of a central processing unit.

Packages: a collection of logically related program entities that are grouped together. Packages provide the means for treating a collection of program entities as a single unit.

Pointers: the use of variables to point at an object in a computer's memory.

Program: a sequence of instructions for performing some operation or solving some problem by computer.

Software: instructions, programs, or data, that enables the computer to do useful work.

Tasking: the capability to execute program entities in parallel (at the same time).

Tools: a variety of software programs that help with the development of a computer program.

VHSIC: a DoD sponsored technology effort to escalate the insertion of advanced microelectronic technology into weapon systems (Very High Speed Integrated Circuits).

BIBLIOGRAPHY

1. Ada's Growth Brings New Techniques and Needs for New Tools." Aviation Week & Space Technology, July 11, 1988, pp. 65-66.

2. Aharonlan, Gregory, "Ada Software Reuse Tool." Defense Computing, September-October 1988, pp. 17-18.

3. Anderson, Chris, "The CAMP Approach to Software Reuse." Defense Computing, September-October 1988, pp. 25-29.

4. Barnes, John and Whitby-Strevens, Colin, "High-Performance Ada Using Transputers." Defense Computing, September-October 1988, pp. 45-49.

5. Bassman, Michell J. and Converse, Robert A., "Ada for System Upgrades." Signal, April 1988, pp. 79-87.

6. Berard, Edward V., and Crafts, Ralph E., "Ada Wars II, Management Is Key To Winning." Defense Science & Electronics, March 1985, pp. 44-52.

7. Boehm, Barry W. Software Engineering Economics. Englewood Cliffs: Prentice-Hall, Inc., 1981.

8. Booch, Grady, "Ada Scores in the International Market." Defense Computing, September-October 1988, pp. 19-24.

9. Canan, James W., "The Software Crisis." Air Force Magazine, May 1986, pp. 46-52.

10. Castor, Virginia L., "Dramatic Progress." Defense Science & Electronics, March 1986, pp. 53-56.

11. Castor, Virginia L., and Preston, David, " Programmers Produce More With Ada." Defense Electronics, June 1987, pp.165-172.

12. Dangerfield, Joseph, Defense Science & Electronics, Interview, December 1986, pp 30-39.

13. Electronic Systems Division Technical Report 86-282, Program Office Guide to Ada Edition I. Hanscom AFB: Department of the Air Force, September 17, 1986.

14. Fostel, Gary, "Just Another Programming Language?" Communications of the ACM, April 1987, pp. 280-281.

15. "General Dynamics Explores Ada In Extensive Flight Test Program." Aviation Week & Space Technology, March 28, 1988, pp. 73-75.

16. Gerhardt, Mark, "Don't Blame Ada." Defense Science & Electronics, August 1987, pp. 53-68.

17. Gray, Lewis, "Pointer to a Point." Communications of the ACM, June 1987, p. 474.

18. Gumble, Bruce, "Getting to Know Ada." Defense Computing, September-October 1988, pp. 32-37.

19. Hall, Major Marilon D., USAF, General Purpose Computers Embedded in Systems, Air Command and Staff College Research Report No. 0885-79, Air University, ATC, Maxwell AFB AL, May 1979.

20. Healy, Kathleen, "Name, Rank and Computer Log-On." Forbes, April 20, 1987, pp. 87-88.

21. Hughes, David, "Next-Generation Defense Programs Will Increase Use of Ada Language." Aviation Week & Space Technology, March 28, 1988, pp. 60-61.

22. "IBM Forges Links to Ada Vendors To Enhance Role in Aerospace Market." Aviation Week & Space Technology, March 28, 1988, pp. 67-69.

23. Johnson, Philip I. The Ada® Primer. New York: McGraw-Hill, 1985.

24. Klein, Daniel, "A buyer's Guide to Ada Procurement." Defense Electronics, January 1985, pp.101-112.

25. Lieblein, Edward, " The Department of Defense Software Initiative - A Status Report." Communications for the ACM, August 1986, pp. 734-744.

26. Litke, John D. and Benedict, Powell A., "A Reusable Ada Library." Defense Computing, September-October 1988, pp. 50-53.

27. Lorenz, Susan, "Ada Shock: A Computer Cultural Transition." Signal, May 1987, pp. 147-156.

28. Nordwall, Bruce D., "Government Agencies Promote Business, Industry

Access to Ada." Aviation Week & Space Technology, November 16, 1987, pp. 91-93.

29. Marlow, William, "There's More to Puzzle." Defense Science & Electronics, March 1986, pp. 38-39.

30. Marshall, Charles, "Ada and Hoppers - A review of MILCOMP'87." Asian Defence, December 1987, pp. 76-81.

31. Miller. Bill, "Ada Powers the Hellfire Missile Program." Defense Computing, September-October 1988, pp. 43-44.

32. Myers, Edith D., "What the Countess Didn't Count On." Datamation, February 1, 1987, pp. 32-36.

33. Office of the Under Secretary of Defense for Acquisition. "Report of the Defense Science Board Task Force on MILITARY SOFTWARE." Washington, D.C., September 1987.

34. Parrish, Liz, "Running in Real Time: A Problem for Ada." Defense Computing, September-October 1988, pp. 38-40.

35. Parris, Scott W. and Olsen, Eric, "The Economics of Ada." Defense Science, February 1988, pp. 41-43.

36. Pinter, Major Michael W., USAF, Computers in Weapons Systems: A Look at the F-15, Air Command and Staff College Research Report No. 88-2135, Air University, ATC, Maxwell AFB AL, April 1988.

37. Price, David. Introduction to Ada. Englewood Cliffs: Prentice-Hall, Inc., 1984.

38. Roske, Tim; Chew, Mark A.; and Pope, Gregory M., "Ada Goes Operational." Signal, April 1988, pp. 31-37.

39. Roy, Daniel M., and Jaworski, Allan, "Birth of a Programming Language." Aerospace America, July 1988, pp. 10-14.

40. Sammet, Jean E., "Why Ada® is Not Just Another Programming Language." Communications for the ACM, August 1986, pp. 722-732.

41. Scott, William B., "Air Force/Navy Test Prove Viability of Ada Use in Flight Critical Systems." Aviation Week & Space Technology, August 29, 1988, p. 16.

42. Shumate, Kenneth C. Understanding Ada. New York: Harper & Row, 1984.

43. "Software." Aerospace America, December 1987, p. 43.

44. Stringfellow, Stan and Sherman, Bruce, "Achieving Ada Code Portability." Defense Science & Electronics, October 1986, pp. 33-41.

45. Suydam, Bill, "AJPO Transfer Draws Mixed Reviews." Defense Computing, September-October 1988, pp. 13-16.

46. Thall, Richard M., and Simpson, Richard T., "Software Systems Development With the Ada Language System." Defense Systems Review, January 1985, pp. 40-43.

47. Torrance, Ca. Reifer Consultants Inc. Donald J. Reifer, "Ada's Impact: A Quantitative Assessment," September 10, 1987.

48. Voelcker, John, "Ada: from promise to practice?" IEEE Spectrum, April 1987, pp. 44-49.

49. Wackwitz, Ronald C., "Ada Alternatives." Datamation, September 1, 1984, p. 15.

50. Washington, D.C. Department of Defense. Office of the Under Secretary of Defense. Ada Joint Program Office. "Ada® Program," April 1986.

51. Washington, D.C. Department of Defense. Office of the Under Secretary of Defense. Colonel Joseph S. Green, "Breaking Barriers."

52. Washington, D.C. Department of Defense. Office of the Under Secretary of Defense. Dr. George P. Millburn, "Department of Defense Statement on the Science and Technology Program," March 1988.

53. Washington, D.C. Department of Defense. Office of the Under Secretary of Defense. Virginia L. Castor, "Ada in U.S. Industry," May 19, 1988.

54. Washington, D.C. Department of The Air Force. HQ USAF." Ada Matur ity and Implementation," November 1988.

55. Washington, D.C. Department of The Air Force. Office of the Special Assistant for Reliability and Maintainability. Major Sue Hermanson, "Evaluation of SAF/AADO Talking Paper on Standard Systems Programming,"

August 1988.

56. Wolfe, Alexander, "New Products Keep Ada Rolling." Electronics, December 2, 1985, p. 20.

57. Wolfe, Alexander, "The Pentagon Unveils Aid for Ada." Electronics, November 27, 1986, p. 31.

58. Young, Stephen J. An Introduction to Ada. West Sussex: Ellis Horwood Limited, 1983.