Volume II

Ada Implementation Guide

March 1992

DEPARTMENT OF THE NAVY

Naval Information Systems Management Center
Space and Naval Warfare Systems Command

REPORT DOCUMENTATION PAGE

Form Approved
OPM No. 0704-0188

1. AGENCY USE ONLY *(Leave Blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	March 1992	Final

4. TITLE AND SUBTITLE
Ada Implementation Plan: Department of the Navy. Volumes [redacted] II

5. FUNDING NUMBERS
N/A

6. AUTHOR(S)
Naval Information Systems Management Center
Space and Naval Warfare Systems Command

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Information Systems Management Center
Space and Naval Warfare Systems Command

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Department of Navy
Naval Information Systems Management Center
Washington, DC 20360-5000

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Unlimited Distribution

12b. DISTRIBUTION CODE
Unlimited

13. ABSTRACT *(Maximum 200 words)*

This two-volume document discusses the use of Ada for software system development in the DoN. It contents apply to both AIS and MCCR communities. The DoN prepared this report ot provide guidance to Navy Program Managers and their staffs as they implement the (1) DoD Appro priations Act, 1992 Public Law 02-172 (Nov 26, 1991), 105 Stat. 1188-1189 and (2) ASN (RDA) memo, Interim Department of the Navy Policy on Ada of 24 June 91 (NOTAL).

14. SUBJECT TERMS
Ada programming language, Navy, systems development, software development, software engineering, Ada9X, policy

15. NUMBER OF PAGES
124

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	Unclassified

Volume II

Ada Implementation Guide

March 1992

DEPARTMENT OF THE NAVY

Naval Information Systems Management Center
Space and Naval Warfare Systems Command

Contents

VOLUME II

List of Tables and Figures

Appendix A
HELPFUL SOURCES

This appendix provides sources to help the Department of the Navy (DON) Program Manager become knowledgeable about Ada-related issues. Information is provided on Government sources such as the Ada Information Clearinghouse (AdaIC), which is sponsored by the Ada Joint Program Office (AJPO) and other sources. The sources listed are not exhaustive, and the information regarding these sources may have changed since the publication of this document. DON does not endorse these sources and Department of Defense (DOD) use of the Ada programming language does not imply in any manner that the DOD endorses or favors any commercial Ada product. These products are listed to inform Program Managers of what is available. Program Managers must use their own judgment about the value of the services. Additional sources can be added to this list for future editions of this guide by contacting the DON Ada Representative.

A.1 GOVERNMENT SOURCES

Government sources are organized into seven categories: organizations, training, publications, bulletin boards, repositories, conferences and special interest groups, and operational development support tools. The type of information contained in each of the categories is as follows:

- **Organizations**—DOD, DON, and Marine Corps organizations that focus on Ada policy, technical guidance, and programs with DON-wide applicability
- **Training**—sources of training and information about training for various types of personnel
- **Publications**—sources of newsletters and other publications
- **Bulletin Boards**—sources that maintain a public bulletin board directed at the Ada community
- **Repositories**—sources of reusable components and libraries
- **Conferences and Special Interest Groups**—information on regularly scheduled expositions, workshops, and symposia as well as conferences
- **Operational Development Support Tools**—information on environments and tools currently used by the DOD community

The amount of Ada-related information available from these sources is too vast to reproduce in this appendix. However, an address, phone number, electronic mail address, and short description of the source are provided. Often, the easiest way to obtain information from these sources is via electronic mail, but contact via mail, telephone, or facsimile also is possible.

A.1.1 Organizations

Ada Joint Program Office
1211 South Fern Street
Room C107
Arlington, VA 22202
(703) 614-0209
DSN 224-0208

AJPO, which consists of a deputy director from each Service and a chairman, is responsible for managing the effort to implement, introduce, and provide life-cycle support for the Ada programming language. The AJPO sponsors AdaIC, a primary source of Ada information.

Ada 9X Project
Project Manager
WL/MNAG
Eglin AFB, FL 32542-5434
(904) 882-8264
anderson@uv4.eglin.af.mil

This project is responsible for revisions to the American National Standards Institute/Military Standard (ANSI/MIL-STD)-1815A to reflect current essential requirements with minimum negative impact and maximum positive impact on the Ada community.

Ada Board
Ada Joint Program Office
1211 South Fern Street
Room C107
Arlington, VA 22202
(703) 614-0209

The Ada Board provides AJPO with balanced advice and information on the technical aspects related to official interpretations of the Ada language standard and on issues associated with Ada validation and software environment activities.

Ada Information Clearinghouse
AdaIC
c/o IIT Research Institute
4600 Forbes Boulevard
Lanham, MD 20706-4320
(703) 685-1477

The clearinghouse produces a quarterly newsletter, which includes special Ada features, questions and answers, letters to the Ada validation organizations, listings of validated compilers, Ada usage database, Ada calendar, letters from the Director, and a bulletin board. (See Section A.2 for directions on how to use the information clearinghouse and for a description of what is in the directories.)

Ada Validation Office
Institute for Defense Analyses
1801 North Beauregard Street
Alexandria, VA 22311
(703) 845-5501

This office implements compiler validation policy and oversees development of the Ada Compiler Validation Capability (ACVC).

National Technical Information Service
Software Standards Validation Group
Building 225, Room A-266
Gaithersburg, MD 20899
(301) 975-3247

The National Technical Information Service (NTIS) is a self-supporting agency of the U.S. Department of Commerce that provides free publications and directories of Government databases and software components.

DON Software Executive Official (SEO)
Commander, Naval Information System Management Center
Building CP5, Room 334
Jefferson Davis Highway
Arlington, VA 22203
(703) 602-2103

This office is the point of contact for all DON software and software-related issues.

DON Ada Representative
AST Software and Systems
Naval Information System Management Center (NISMC)
Building 166, Washington Navy Yard
Washington, D.C. 20374
(202) 433-4903/3499

This office is the point of contact for all Ada and Ada-related issues.

Space and Naval Warfare Systems Command (SPAWAR)
Code 224-1
5 Crystal Park
Suite 700
Washington, D.C. 20363-5100
(703) 602-9188

This office is the point of contact for Tactical Digital Standards (TADSTANDs) and Mission-Critical Computer Resources (MCCR) waiver processing.

Commander, Naval Computer and Telecommunications Command
(COMNAVCOMTELCOM)
Ada Program Manager
4401 Massachusetts Avenue, NW
Washington, D.C. 22036-5000
(202) 282-2563
DSN 292-2563
FAX (202) 282-2563

This Command is the headquarters of the Naval Computer and Telecommunications Stations (NCTSs). Through the Ada Technical Support Bulletin Board, NCTC provides support for Ada projects performed by the NCTS activities specifically and the Ada technical community at large. NCTC chairs the AdaSAGE Configuration Management Board, oversees the DON Reusable Ada Products for Information Systems Development (RAPID) site, and publishes CHIPS.

Commandant of the Marine Corps
Mission-Critical Computer Resources (MCCR) Policy Officer
Headquarters U.S. Marine Corps
Code C2IC
Washington, D.C. 20380
(703) 614-8780
DSN 224-8780
FAX (703) 697-1959

As the point of contact for Marine Corps MCCR policy, this source can provide information pertaining to all Marine Corps Ada developments, programming environments, and education.

Commandant of the Marine Corps
Director
Marine Corps Computer and Telecommunications Activity
Headquarters and Service Battalion
Building 2006
Marine Corps Development and Education Command
Quantico, VA 22134-5001
(703) 640-4721
DSN 278-4721

As the point of contact for Marine Corps Automated Information System (AIS) Ada policy, this source can provide information pertaining to all Marine Corps Ada developments, programming environments, and education.

Software Engineering Institute
Carnegie-Mellon University
Pittsburgh, PA 15213
(412) 268-7700

The Software Engineering Institute (SEI) was established in December 1984 at Carnegie-Mellon University (CMU) as a Federally Funded Research and Development Center (FFRDC). The SEI is sponsored by the DOD and managed by the Defense Advanced Research Projects Agency (DARPA) through a joint program office at the Electronics Systems Division (ESD) of the Air Force Systems Command (AFSC). The SEI mission is to provide leadership in advancing the state of the practice of software engineering to improve the quality of systems that depend on software. Numerous free publications are available on Ada-related issues from the SEI.

Software Technology for Adaptable, Reliable Systems
1500 Wilson Boulevard
Suite 317
Arlington, VA 22209
(703) 243-8655
E-mail address for newsletter: newsletter@stars.rosslyn.unisys.com.
E-mail address for STARS Repository: blanchard@stars.startech.com.

Software Technology for Adaptable, Reliable Systems (STARS) is a Government agency that is deeply involved with software. The software project goals cited in the STARS charter are as follows:

- Improve productivity
- Improve quality and reliability
- Promote development and applications of reusable software
- Reduce time and cost of developing defense software

STARS has used four thrusts to achieve these goals:

- STARS foundation contracts
- STARS competing prime contracts
- STARS Ada Shadows contracts
- STARS Workshop

This Government-sponsored program is attempting to develop the foundation for a coordinated DOD software solution based on three integrating elements:

- A set of Software Engineering Environments (SEEs)
- A set of tailorable software life-cycle process building blocks
- A software asset library capability

Emphasis is on commercially supported products based on modern process models with open interfaces and reusable software libraries.

STARS maintains an affiliates' program to provide an opportunity for the DOD software community to participate in STARS technical activities. Affiliates are individual representatives of Government agencies, universities, vendors, etc. The two levels of affiliates are as follows:

- Information Exchange Affiliates play an active role in STARS programs; have access to information, such as newsletters and bulletins; and participate in briefings and working groups.

- Prime Affiliates work directly with STARS prime contractors in relevant technical activities such as technology transition, product evaluation, and tool development.

Labor, travel, and other expenses associated with participation in the affiliates program are the responsibility of the parent sponsoring organization.

A.1.2 Training

Ada Language System/Navy
ALS/N Training
Fleet Combat Direction System Support Activity
San Diego, CA 92147-5081
POC: Code 8D
(619) 553-4440

Training provided at this source includes Ada Language System/Navy (ALS/N) courses pertaining to Ada/L (UYK-44 target), Ada M (AN/UYK-43), PPI (AN/AYK-14 target), and Common Ada Baseline/Project Support Environment (CAB/PSE) tools (VAX/VMW host with associated PSE tools). Training is conducted quarterly in conjunction with ALS/N public reviews, and on-site training is available.

Ada Software Engineering Education and Training Team
Institute for Defense Analyses
1801 North Beauregard Street
Alexandria, VA 22311
Attn: Resource Staff Member
(703) 845-6626

The Ada Software Engineering Education and Training (ASEET) Team is composed of representatives from the Army, Air Force, Navy, Marine Corps, other DOD agencies, and academia. The team conducts workshops and symposia for Ada educators within DOD and academia and coordinates the activities of DOD organizations engaged in meeting the Ada education and training needs. An ASEET resource library of educational materials is located at AJPO.

The team has tri-Service representation on four working groups that address education and training on requirements, courseware, professional development, and coordination. Major tasks include the following:

- Identify education and training requirements within DOD
- Conduct Ada research projects
- Manage Ada course materials
- Perform Ada certification study
- Provide a database of ASEET research data and final reports
- Provide a DOD focal point for Ada software engineering education and training

AdaSAGE
Department of Energy
Idaho National Engineering Laboratory (INEL)
Idaho Falls, Idaho
(208) 526-0656

INEL offers training and support via a Hot Line subscription service.

Application Programming Instructional Department
Computer Science School
Marine Corp Combat Development Command
Quantico, VA 22134-3554
(703) 640-2962
DSN 278-3554/2962

The Application Programming Instructional Department is responsible for the C curriculum for the DON Data Processing rating.

Common APSE Interface Set
CAIS-A
Commander
Naval Ocean Systems Center
271 Catalina Blvd
San Diego, CA 92152-5000
Attn: CAIS-A Training Coordinator
(619) 553-6858

A 5-day training class is available that provides hands-on experience for Ada tool designers. (Knowledge of the Ada programming language is a prerequisite.) Training will be available on a VMS system and (UNIX) SUN 3. Also available are the following:

- CAIS-A Self study guide
- CAIS-A Tool writers guide

Computer Sciences School
Head Applications Programming Instructional Department
Marine Corps Combat Development Command (MCCDC)
Quantico, VA 22134
(703) 640-2962
DSN 278-2962

The Computer Sciences School (CSS) currently offers two 1-month (20 training days) Ada programming courses per year; beginning in FY 92, three courses per year will be offered. Upon completion, the trainee is qualified as a basic Ada programmer. The course has been offered primarily to active-duty Marines and civilians employed by the Marine Corps, but the Navy has shown interest in using the training, starting in FY 92. Active-duty or reserve Marines and civilians employed by the Marine Corps who are interested in enrolling in this training should contact Mr. Steve Bruzek, Headquarters U.S. Marine Corps at DSN 224-1886 or Comm (703) 614-8780. Active-duty or reserve Navy personnel should contact the Navy DP Detailer, NMPC-406, DSN 223-3537 or Comm (703) 693-3537. Civilians employed by the Navy should contact the CSS Academics Office, (703) 640-2891 or DSN 278-2891.

National Audiovisual Center
8700 Edgeworth Drive
Capitol Heights, MD 20743-3701
Attn: Customer Service Department
(301) 763-1891
FAX (301) 763-6025

A series of Ada training tapes sponsored by the AJPO is available for purchase through the National Audiovisual Center of the Department of Commerce. The tapes include the following:

- Introduction to Ada (3 tapes; about 3 hours total; order # A18336; $150)
- Ada from a Management Perspective (2 tapes; about 80 minutes; order # A18337; $100)
- Software Engineering in Ada (19 tapes; about 8 hours, 20 minutes; order # A18338; $500)

Additional information on these tapes is available from the AdaIC.

United States Army Engineering College
Rock Island Arsenal
Rock Island, IL 61299-5000
Attn: AMXOM-PMR
(309) 782-0488/0489/0487
Course No. AMEC-140

The U.S. Army Engineering College provides a 2-week Ada overview free to Government employees.

* * * * *

All of the major Ada compiler vendors have training available directly through their offices.

A.1.3 Publications

Ada and C++
STSC, OOALC/TISAC,
Hill Air Force Base (AFB), UT 84056
Attn: Gary Peterson
(801) 777-7703
DSN 458-7703

This report describes studies that compared Ada to C++. An electronic summary of this report is available on the AdaIC Bulletin Board.

Ada Information Clearinghouse Newsletter
c/o IIT Research Institute
4600 Forbes Boulevard
Lanham, MD 20706-4320
1-800-Ada-ICII or (703) 685-1477

The AdaIC publishes a quarterly newsletter containing information on the Ada community's events, working groups, research, publications, and concerns. The clearinghouse provides its services free of charge to Government, academic, and commercial software communities.

Ada Slices
MITRE Corporation
1120 NASA Road 1
Houston, TX 77057
(713) 335-8541

This newsletter is published by MITRE, a FFRDC. It is a product of the Association for Computing Machinery (ACM) the Special Interest Group on Ada's (SIGAda's) Performance Issues Working Group (PIWG) and is available free of charge.

Ada Software Engineering Education and Training Public Report
Ada Software Engineering Education and Training Team
Institute for Defense Analyses
1801 North Beauregard Street
Alexandria, VA 22311
Attn: Resource Staff Member
(703) 845-6626

ASEET publishes a DOD ASEET Public Report annually. The report contains an update and description of the latest efforts of the ASEET Team in identifying training and education requirements within DOD and the methodology and materials needed to fulfill those requirements. Copies of the report may be obtained from the AdaIC.

Bridge
Linda Pesante
Managing Editor, Bridge
Software Engineering Institute
Carnegie-Mellon University
Pittsburgh, PA 15213-3890
(412) 268-7622
bridge-editor@sei.cmu.edu

This magazine reports on SEI projects and activities. To obtain a subscription, send a written request to the editor.

C2MUG Bulletin
C2MUG Catalog
Department of the Army
Chief CECOM SEDPMS
Building 138
Fort Leavenworth, KS 66027-5600
Attn: AMSEL-RD-SE-MCS-MC (C2MUG)
(913) 684-7550
DSN 552-7550
amsel-rd-se-bcs-mc@monmouth-emh2.army.mil

C2MUG Bulletin is the bimonthly newsletter of the Command and Control Microcomputer Users' Group. The C2MUG Catalog is a semi-annual publication that lists more than 500 military applications, Ada programs, and general utilities.

Crosstalk
Software Technology Support Center
Ogden Air Logistics Center
TISAC
Hill AFB, UT 84056
(801) 544-9200
DSN 458-7703

Crosstalk is the Software Technology Support Center (STSC) newsletter. STSC is the Air Force Office of Advocacy and the focal point for the development, evaluation, and promotion of compatibility in software tools, methods, and environments for mission-critical computer sources.

DACS Newsletter
Data & Analysis Center for Software
258 Genesee Street
Suite 101
Utica, NY 13502
(315) 336-0937

Published since 1982, this quarterly is the newsletter of the Data and Analysis Center for Software (DACS), Griffiss AFB.

Defense Technical Information Center
Cameron Station
Alexandria, VA 22304
(703) 274-7633

The Defense Technical Information Center (DTIC) distributes documents only to military, Government, or defense contractors who are registered users of DTIC. All unclassified documents that are submitted to DTIC are automatically forwarded to NTIS and made available to the public.

Institute for Defense Analyses
Computer & Software Engineering Division
1801 North Beauregard Street
Alexandria, VA 22311
(703) 845-2000 (General)
(703) 845-2059 (References)

The institute is an FFRDC whose primary sponsor is the Office of the Secretary of Defense. All of the publications prepared by the institute are available through DTIC or NTIS.

Language Control Facility Ada-JOVIAL Newsletter
ASD/SCEL
Wright-Patterson AFB, OH 45433-6503
(513) 255-4472
langed@wpafb-jalcf.arpa

To support the DOD and Air Force standardization efforts, information is disseminated about Ada and JOVIAL (J73), standardization and language control activities, training, compilers, compilers and tools, development efforts, applications, and user services.

National Standards Association
Bethesda, MD 20816
(301) 590-2300

The National Standards Association is an alternative source for ordering military standards, specifications, handbooks, etc. There is a charge for the service. Documents are shipped within 24 hours of order. ($10 per item up to 50 pages; additional pages cost $.20 each)

National Technical Information Service
U.S. Department of Commerce
5285 Port Royal Road
Springfield, VA 22161
(703) 487-4650

The NTIS is sponsored by the U.S. Department of Commerce and has over 2 million publications available to the public.

NISMC Newsletter
Ms. Alcinda Wenberg
NISMC
Building CP5
Jefferson Davis Highway
Arlington, VA 22203
(703) 602-2542

This monthly newsletter provides information on NISMC initiatives, status of DON policy, and upcoming DON activities.

STARS Newsletter
1500 Wilson Blvd.
Suite 317
Arlington, VA 22209
(703) 243-8655
newsletter@stars.rosslyn.unisys.com

The STARS newsletter contains articles covering software reuse technology, software process technology, and software engineering environment framework technology. It is published three times per year.

Standardization Documents Order Desk
Building 4, Section D
700 Robbins Avenue
Philadelphia, PA 19111-5094
Special Assistance Desk—(215) 697-2179; DSN 442-2179
Customer Service—(215) 697-2667

This desk is the central distributor of all military standard documents, including the standard for the Ada language reference manual (ANSI/MIL-STD-1815A-1983). To contact a customer representative, call (215) 697-2179 or DSN 442-2179. DOD standards, specifications, handbooks, and data items can be ordered by using the Telephone Order Entry System (TOES). Access TOES by calling (215) 697-1187 (DSN 442-1187), Monday through Friday, 8:00 a.m. to 6:00 p.m. e.s.t.

Written requests must be submitted for ANSI documents and DOD instructions and directives. Only two copies of the same item are sent, and there is no bulk mailing.

A.1.4 Bulletin Boards

Ada 9X Project
Project Manager
WL/MNAG
Eglin AFB, FL 32542-5434
(904) 882-8264
anderson@uv4.eglin.af.mil

Information can be obtained from the ADA 9X BULLETIN BOARD by calling 1-800-Ada-9X25 or (301) 459-8939 or by using the electronic address shown above. Baud rates of 300/1200/2400 will work on the system. To access by modem, use the following settings:

- Baud rate = 300, 1200, or 2400
- Parity = none
- Data bits = 8
- Stop bits = 1

Ada Joint Program Office
1211 South Fern Street
Room C107
Arlington, VA 22202
(703) 614-0209

The AJPO sponsors a bulletin board called the AdaIC Bulletin Board. This bulletin board is a primary source for Ada information. Access is available by modem or through the Internet to the ajpo host. Section A.2 describes this important bulletin board in detail.

Ada Language System/Navy Bulletin Board System
(202) 342-4568

The ALS/N has a bulletin board system available for interested Ada users. Important dates, releases, and policies are posted on the system. Also, this bulletin board enables users to exchange information on lessons learned.

Use the following settings to access the bulletin board by modem:

- Baud rate = 300, 1200, or 2400
- Parity = none

- Data bits = 8
- Stop bits = 1
- Parameters = full duplex

If additional information is needed, please contact Greg Engledove, Naval Sea Systems Command, (703) 602-8204.

Ada Technical Support Bulletin Board Service

Naval Computer and Telecommunications Area Master Station Atlantic
(NCTAMS LANT)
Norfolk, VA
(804) 444-7841
Autovon: 564-7841

To access by modem, use the following settings:

- Baud rate = 300, 1200, or 2400
- Parity = none
- Data bits = 8
- Stop bits = 1

The NCTC sponsors an Ada Technical Support Bulletin Board System (BBS) maintained by NCTAMS LANT.

The main purpose of the BBS is to offer DOS-Ada programmers in the joint Services and Government contractors a means for obtaining answers to their questions about the Ada programming language. The BBS is targeted to programmers in the AIS domain on 80x86 DOS-based systems and to software creation on these systems.

The BBS offers several services:

- Ada Question and Answer Service—BBS users can ask questions about the Ada language and extensions (e.g., pragmas) that might be included in a particular implementation. Additionally, user code can be uploaded for evaluation. Such evaluations can include checks for proper usage of Ada features, Ada style, and compilation errors that will not go away.

- Compiler Vendor Comment Service—BBS users can comment on DOS-based Ada implementations. Comments can report either problems with existing implementations or suggest enhancements that would benefit the DOS-Ada community. These comments will be provided to the appropriate compiler

vendors. The goal is to use this service to improve DOS-based Ada compilers. A secondary benefit is to make potential users aware of possible problems with particular DOS-based Ada compiler implementations.

- Ada Limited Debugging Assistance—BBS users can upload small amounts of code to be debugged. The submitted code must be limited to a few program units.

- AdaSAGE Question and Answer Service—Many DOS-Ada application developers use AdaSAGE for database management system (DBMS) functions. This service is for AdaSAGE users. Users will be able to ask one another questions about AdaSAGE.

- AdaSAGE Comment Service—BBS users can comment on Ada application development using AdaSAGE. Comments can either report problems with AdaSAGE or suggest enhancements that would benefit future versions of AdaSAGE. These comments will be collected and presented periodically at AdaSAGE enhancement meetings. Users may also request AdaSAGE enhancements.

- Ada Example Set—This collection of code shows Ada features. BBS users can download the code, study it, and ask questions about it. Users also can upload code that shows Ada features.

- News—The BBS will list Ada news, events, and interesting Ada products with their points of contact.

The service is free and available to the public. However, the limited debugging service is available to bona fide Government employees and their contractors.

A.1.5 Repositories

Ada Software Repository
ada-sw-request@wsmr-simtel20.army.mil

Ada Software Repository (ASR) is a repository of Ada programs, software components, and educational material that has been established on the Defense Data Network (DDN). This repository has been accessible to any host computer on the DDN since November 26, 1984.

ASR provides a free source for Ada programs and information. By employing the File Transfer Protocol (ftp) program, users of DDN hosts are able to scan the

directories of the repository and transfer files to their hosts. If the files are Ada programs, they may then compile these programs and use them as they desire. Modifying these programs may be within their rights, and they may freely distribute these programs as they desire, subject to the restrictions specified for each piece of software in its prologue.

All members of the Ada community are encouraged to freely extract information and programs from the repository and to make contributions to it. The only restrictions that apply to access and use of this software are presented in the "Distribution and Copyright" section of the prologue associated with each piece of software.

ASR is one of several repositories located on the SIMTEL20 DDN host computer at White Sands Missile Range in New Mexico. SIMTEL20 is owned and operated by the Operations and Systems Integration Division of the Information Systems Command of the U.S. Army.

ASR maintains source code from approximately 10,000 Ada programs. These programs are maintained by domains of Artificial Intelligence, Benchmarks, Communications, Reusable Software Components, Database Management, Documentation, Graphics, Project Management, Ada Software Development Tools, and many others. The ASR is available via ftp and on magnetic tape, floppy disk, and CD-ROM.

An introduction to the ASR can be obtained by using the following commands on a system that supports ftp on the DDN:

```
>ftp wsmr-simtel20.army.mil
when asked for login name, type in anonymous
when asked for password, type in your user-id
ftp>ls                          —provides listing of login directory
ftp>get SIMTEL20-ADA.INF        —copies file to your local directory
ftp>quit                        —returns control to UNIX
```

Tape copy is available from:
The DECUS Program Library
219 Boston Post Road BP02
Marlboro, MA 01752
(508) 480-3418

MS-DOS high-density diskette copy is available from:
Advanced Software Technology, Inc.
P.O. Box 937
Medford, NY 11763
(516) 289-6646

CD-ROM copy is available from:
ALDE Publishing
P.O. Box 35326
4830 West 77th Street
Minneapolis, MN 55435
(612) 835-5240
FAX: (612) 835-3401

An electronic mailing list exists on SIMTEL20 for those who are interested in accessing and contributing software to the ASR. To subscribe to this mailing list, send a request to the electronic mail address above.

Associate Director, MCSD
AMSEL-RD-SE-BCS-MC (C2MUG)
Fort Leavenworth, KS 66027
AV: 552-7550
FTS: 753-7550
(913) 684-7550

The C2MUG Software Catalog for mathematics and various Ada functions is available to all echelons of the U.S. military and elements of the Federal Government. Software components are primarily for microcomputers.

Command, Control, Communications, and Intelligence (C3I) Reusable Software System (CRSS)

Mr. Ron Crepeau
NRaD
271 Catalina Blvd.
San Diego, CA 92152-5000
(619) 553-3990
crepeau@nosc.mil

The CRSS is a repository of Navy command and control (C2) assets, including source and executable code, documentation, and graphical representations. The library has been developed under the Operations Support System (OSS) project to promote rapid prototyping of C2 systems. Replication of the CRSS is available on magnetic media or via modem transfer.

Common Ada Missile Components Effort

Data and Analysis Center for Software
c/o Kaman Sciences Corporation
P.O. Box 120
Utica, NY 13503
(315) 336-0937

Common Ada Missile Packages (CAMP) are operational flight software parts in Ada for tactical missiles. CAMP consists of 454 reusable Ada components. The software is distributed on ANSI standard labeled 9-track 1600-b.p.i tapes.

Data and Analysis Center for Software

258 Genesee Street
Suite 101
Utica, NY 13502
(315) 336-0937

Although not an interactive repository, DACS provides several products and services. Of importance are the following: the Ada Compiler Evaluation Capability (ACEC), CAMP, a variety of data sets, software sizing models, and a variety of data collection form sets.

National Aeronautics and Space Administration's AdaNet

AdaNet
c/o MountainNet
Eastgate Plaza, 2nd Floor
P.O. Box 370
Dellslow, WV 26531-0370
(304) 296-1458
1-800-444-1458

AdaNet's primary purpose is to increase U.S. productivity, economic growth, and competitiveness through development of a life-cycle repository for software engineering products, processes, interfaces, and related information services. AdaNet is sponsored by NASA, and there is no charge for an account.

AdaNet provides the following information and services:

- Access to Ada source code libraries
- Bibliographic references to Ada and software engineering publications
- Descriptions of public and commercial repositories of Ada software
- Directories of Ada and software engineering commercial products
- Electronic forums on topics such as software reuse and Computer-Aided Logistics Support (CALS)
- Listings of international Ada professional organizations
- Monthly listings of relevant conferences and seminars
- References to public and private Ada information services

Navy Wide Reuse Center

nhis.navy.mil

The Navy Wide Reuse Center (NWRC), which was dedicated on 16 March 1992, will provide a comprehensive reuse support environment for all Navy domains. The center will serve as a repository for all Navy reusable components and provide interfaces to other DOD and non-DOD repositories as well as information on commercially available reusable components. NWRC uses the Defense Software Repository System (DSRS) hosted on a DEC MicroVax computer. DSRS is

accessible via DDN, modem dial-up, and selected local area networks. An account on the system is required. A request for an account should be made through the following:

Project Manager, Navy Wide Reuse Center
Washington Navy Yard
Building 196
Code N53, Room 4508
Washington, D.C. 20374
POC: Angus Faust
(202) 433-0718

Reusable Ada Products for Information Systems Development
U.S. Army
USAISSDCW
RAPID Center
Fort Belvoir, VA 22060-5456
Attn: ASQBI-SS-WRC Stop H-4
(703) 285-9007
DSN 365-9007

The RAPID Program is a total Ada software reuse program established at the U.S. Army Information System Software Center (ISSC) Software Development Center, Washington (SDC-W). This program has become the basis for the DSRS under the Defense Information Systems Agency (DISA) Corporate Information Management (CIM). Under this system a repository has been established for each service. The NWRC serves as the repository for all Navy reusable components and provides an interface to the central repository and other service repositories under the DSRS.

Software Technology for Adaptable, Reliable Systems Repository
blanchard@stars.startech.com

STARS maintains a repository of Ada binding to MOTIF, Ada binding to Ada Xt Windows Intrincints, Reuse Library Framework (RLF), and much more.

Tape copy is available from:
ASEET
2611 Cranberry Square
Bldg. 2600, Suite 2
Morgantown, WV 26505
(304) 594-1762

MS-DOS high-density diskette copy is available from:
Advanced Software Technology, Inc.
P.O. Box 937
Medford, NY 11763
(516) 758-6545

A.1.6 Conferences and Special Interest Groups

Ada Joint (Services) Users Group
P.S.S. Chairperson
(212) 394-5233
Mike Ryer
(617) 661-1840
Cambridge, MA 02138
Loretta Holden (Boeing)
(206) 662-0232

Ada Joint (Services) User Group (AdaJUG) (formerly JOVIAL Users Group) is a non-profit organization whose purpose is to encourage a dialogue between Government and industry concerning Ada issues. The AdaJUG holds conferences three times a year. Generally, at least one of the conferences is held in conjunction with SIGAda. Topics include Ada 9X progress and distinguished Government corner, which usually represents three DOD components. ($25 per meeting)

ASEET Symposium
Institute for Defense Analyses
1801 North Beauregard Street
Alexandria, VA 22311
(703) 824-5531

The ASEET Team Coordinator Working Group (CWG) sponsors the annual ASEET Symposium. The symposium enables DOD personnel to learn about new education and training methods from industry, academia, and DOD organizations.

DON Ada Users Group
DON Ada Users Group-Chair
Naval Ocean System Center
271 Catalina Boulevard
San Diego, CA 92152-5000
Tel: (619) 553-2303
FAX: (619) 553-5799

The DON Ada Users Group has been chartered to provide information and support to the DON on use of the Ada programming language and Ada-related issues associated with software development and maintenance. Regular meetings are held in conjunction with national conferences in the Ada community, such as those sponsored by Tri-Ada, SIGAda, and AdaJUG.

STARS Workshop
IDA/CSED
5111 Leesburg Pike
Falls Church, VA 22041
(703) 845-3520

The STARS Joint Program Office holds workshops to publicize and disseminate information on various contract efforts.

A.1.7 Operational Support Development Tools

Ada Language System/Navy
Naval Sea Systems Command (PMS-412)
NC-3
2531 Jefferson Davis Highway
Washington, D.C. 20362-5101
(703) 602-8204

The ALS/N is a software development environment and Run-Time Environment (RTE) system that is being developed for the current generation of DON standard computers, the AN/UYK-43 and 44 and AYK-14. ALS/N has a users group and a bulletin board available upon request and target-based training available through the project office.

AdaSAGE
Department of Energy
Idaho National Engineering Laboratory (INEL)
Idaho Falls, ID
(208) 526-0656

AdaSAGE is a Government-owned development reuse tool utilized by all DOD components. The tool consists of utilities that support user-developed interfaces, reports, and data and their relationships. These utilities facilitate user-directed rapid prototyping. AdaSAGE is free and is supported by a Joint Service Configuration Management Board. Enhancements may be requested by leaving a message on the Ada Technical Support Bulletin Board listed in A.1.4.

NAVAIR Software Engineering Environment Tool Set
Mr. Ron Conley
Code 7012
Naval Air Development Center
Warminster, PA 18974-5000
(215) 441-2752

The Naval Air Systems Command (NAVAIR) Software Engineering Environment (NASEE) Working Group contracted for 12 software tools for use throughout the software life cycle. NAVAIR has made these tools a standard for its Software Support Activities. The Naval Air Development Center provides information on the NASEE Working Group and on ways to obtain these tools.

Tool Box PC
Software Technology Support Center
Hill AFB, UT 84056
1-800-477-2449

This tool, which is written in Ada, is an interactive catalog application tool that has Ada and other Government and commercially owned software languages. This system is designed for managers to use. The program is available free to the public on 5¼- and 3½-inch disks. The Air Force supports this program through the STSC.

A.2 Ada INFORMATION CLEARINGHOUSE

AdaIC disseminates information on Ada policy, validated Ada compilers, available Ada bindings, and Ada software development tools as well as on the Ada community's events, working groups, research, publications, and concerns. This information is available to anyone who wishes to know more about Ada. Information can be obtained in hard-copy or electronic form from our electronic bulletin board or on the AJPO machine on the DDN. All AdaIC services and products are provided free of charge. The AdaIC service is sponsored by the AJPO, which facilitates the implementation of Ada throughout the Services and maintains the integrity of the language. IIT Research Institute operates the AdaIC for the AJPO.

Hot Line: 1-800-ADA-IC11 or (703) 685-1477
If you have a question about any aspect of using Ada or being part of the community, call the hot line between 8 a.m. and 5 p.m. on Monday through Friday.

Written Inquiries
If you prefer to send a written inquiry or would like to share any Ada-related information with us, send mail to:

AdaIC
c/o IIT Research Institute
4600 Forbes Boulevard
Lanham, MD 20706-4320

On-Line Information
A variety of Ada-related information is available on-line on the AdaIC's electronic bulletin board (703) 614-0215 or (301) 459-3865 and on the AJPO host computer on the DDN. For instructions on how to download files, see the AdaIC flyer, "Public Access to the AdaIC Bulletin Board" (bulletin-board file ada-rbbs.hlp), and "Access to Ada Information on the Defense Data Network" (ada-ddn.hlp).

Databases
Products and Tools—AdaIC maintains a database of currently available products and tools for Ada. To obtain a search, call (703) 685-1477.

Ada Usage—AdaIC maintains a database of Ada projects. If you are currently involved in an Ada development project or have completed a project using Ada, please contact us. We would like to add your information to our database.

Quarterly Newsletter
AdaIC publishes a quarterly newsletter containing current news, features, columns, the most recent Validated Compilers List, and articles on projects using Ada. To receive a copy of the newsletter, please call and request a free subscription.

Informational Flyers
AdaIC regularly updates and publishes more than 70 separate informational flyers and lists that provide information on all aspects of using Ada. Among AdaIC offerings are items such as:

- Ada events calendar
- AJPO, Ada 9X Project Office, and Ada community announcements
- Ada 9X information and copies of the Ada 9X project reports
- Information on Ada usage
- Ada products and tools information
- Electronic resources
- Ada books and documents
- Ada validation information

- Ada performance information
- Ada education
- Ada policies
- Ada historical information

Ada 9X Project Information
AdaIC serves as the distribution point for information related to the Ada 9X Project. Project report announcements and reports from the Ada 9X Project Manager are available electronically from the Ada 9X electronic bulletin board (1-800-ADA-9X25 or (301) 459-8939). This information is also available in the DDN on the AJPO host in the Ada 9X subdirectory.

A.2.1 Public Access to the AdaIC Bulletin Board (ada-rbbs.hlp extract)
The AdaIC Bulletin Board is a publicly available source of information on the Ada language and Ada activities. Sponsored by the AJPO and maintained by AdaIC, this bulletin board is used to announce current events and general activities and provide a current listing of validated Ada compilers. Access to the bulletin board requires a computer terminal and modem or a personal computer and modem.

The AdaIC Bulletin Board system can be accessed by dialing (703) 614-0215 or (301) 459-3865, using a 300-, 1200-, or 2400-baud modem. Users should set their telecommunications package with the following parameters:

- Baud rate = 300, 1200, or 2400
- Parity = none
- Data bits = 8
- Stop bits = 1

Currently, the following 12 directories are available:

- The Ada Information Directory—an alphabetical listing of all available information files, with a contents description for each one

- The Language Reference Manual Directory—the Ada Language Reference Manual (ANSI/MIL-STD-1815A-1983) in its entirety

- The Approved Ada Commentaries Directory—approved commentaries responding to questions, problems, and/or inconsistencies and perceived inconsistencies regarding the Ada Language Reference Manual (ANSI/MIL-STD-1815A-1983)

- The Ada Language Rationale Directory—the rationale for the Design of the Ada Programming Language in its entirety
- The CAIS Document Directory—the Common Ada PSE Interface Set (CAIS) documents (October 1986)
- The AdaIC Newsletter Directory—past AdaIC newsletters
- The CREASE Directory—AJPO'S *Catalog of Resources for Education in Ada and Software Engineering*, version 5.0, in its entirety
- The Miscellaneous Directory—files such as those used to decompress compressed files
- Directories 9 and 10—a guidebook and reference manual, respectively, for the evaluation and validation of Ada programming support environments
- Directory 11—the NASA-Goddard Ada Style Guide, which was proposed as the basis for a military handbook
- Directory 12—a catalog of the ASR and the ASR User's Handbook

Files are available in either compressed or uncompressed (ordinary ASCII text-file) format. Most are available in both.

A.2.2 Access to Ada Information on the Defense Data Network (ada-ddn.hlp extract) The public directory on the ajpo host computer is an official source of information on the Ada language and Ada activities. Sponsored by AJPO and maintained by AdaIC, this computer directory is used to announce current events and general activities and to provide a current listing of validated Ada compilers.

This directory is available only to authorized users of DDN. However, AdaIC also maintains a bulletin board at (703) 614-0215 and (301) 459-3865. For information, see the AdaIC handout, "Public Access to the Ada Information Bulletin Board" (AdaIC from G/V51, file ADA-RBBS.HLP).

The DDN is a collection of approximately 80 different computer networks representing DOD facilities, research centers, and academic institutions throughout the free world. All of the networks are packet-switching systems with

interconnections at various locations. DOD controls access to the DDN. To obtain access to the DDN, it is first necessary to have an account or access to an account on one of the several thousand host computers that make up the system.

The following set of commands provides an example of the use of ftp to transfer a file from the ajpo host to a local host. The file is in the directory public/ada-info/val-comp.hlp.

```
>ftp ajpo.sei.cmu.edu
when asked for login name, type in anonymous
when asked for password, type in your user-id
ftp>ls                      —provides listing of login directory
ftp> cd public              —changes directory to the public directory
ftp> cd ada-info            —changes directory to the ada-info directory
ftp> get val-comp.hlp       —copies file to your local directory
ftp> quit                   —returns control to UNIX
```

As of 17 March 1992, directories in the public directory include acvc-current, ada-adoption-hbk, ada-comment, ada-info, ada-lsn, ada-ui, ada9x, adanews, adastyle, artdata, asis, cais, crease50, ev-info, infoada, kitdata, lrm, pcis, piwg, rationale, wbs.sw. These files correspond to those shown in figure A-1. The primary Ada information files are provided in the AdaIC File Directory in this appendix.

A.2.3 Info_Ada Digest

DDN users can also access the Info_Ada Digest (to send discussions to the digest, use info_ada@ajpo.sei.cmu.edu). To request that you be added to the discussion list, use info_ada_requests@ajpo.sei.cmu.edu. Alternatively, the same discussions are available via USENEWS news group comp.lang.ada.

DDN users can also access the Ada_Ed Digest. To send discussions, use ada-ed@east.pima.edu; to request that you be added to the discussion list, use ada-ed-requests@east.pima.edu.

The Ada electronic mailing list includes the following:

- Ada announcements
- Open forum for discussion
- Open forum for questions
- Requests for information to the entire Ada community

A.2.4 Document Reference Sources

In addition to the information available from AdaIC, many documents are available from sources described below. This information is taken from the AdaIC Document Reference List.

Government Printing Office

Superintendent of Documents
Government Printing Office
Washington, D.C. 20402
(202) 783-3238

The Government Printing Office (GPO) distributes the Reference Manual for the Ada Programming Language to the general public and industry for $16 a copy. Mail orders may be sent to the above address with payment included. Phone orders are accepted with a VISA or Master Card number or a GPO deposit account number. For additional information, call the number noted above.

Government Source Codes

SEI= Software Engineering Institute
AJPO = Ada Joint Program Office
WPAFB = Wright Patterson Air Force Base
CECOM = Communications Electronic Command
USAF = United States Air Force

A.2.5 AdaIC File Directory

The information in Figure A-1 was listed in the AdaIC Bulletin Board in March 1992. It details the types of information available from AdaIC.

Figure A-1. AdaIC Directories

Directory Numbers and General Description of Contents*

1 Ada Information Files
2 Language Reference Manual
3 Approved Ada Commentaries
4 Ada Language Rationale
5 CAIS Document
6 AdaIC Newsletter
7 Catalog of Resources for Education (CREASE)
8 Miscellaneous - Unzipping Utilities
9 APSE E&V Guidebook V2.0
10 APSE E&V Reference Manual
11 Proposed Ada Style Guide
12 ASR User's Handbook & Directory

* To list Directory, type l:1
For a list of all available files on the system, download director.zip

AdaIC Information Files—Directory 1
This directory contains electronic copies of the flyers and other documents offered by AdaIC. In addition, it contains electronic copies of several DOD directives relating to the Ada programming language.

The files below are listed with the extension ".HLP". When you use the download command, you will be prompted for the filename. If you give the filename with the .HLP extension, you will get an ordinary ASCII text file. However, to reduce the time required for downloading to your computer, most of the files listed below are also available in compressed (ZIPped) format. To download a file in compressed format, substitute .ZIP for the .HLP extension.

To view these ZIPPED files, you need an unzipping utility, which is available on this and many other bulletin boards. (See Directory 8 and Bulletin 2.)

File Name	Updated	Size	Contents
3405-1	7/18/89	18644	Text of 4/2/87 DoD Directive 3405.1, Computer Programming Language Policy
3405-2	7/10/89	7709	Text of 3/30/87 DoD Directive 3405.2 mandating use of Ada language in computers integral to weapon systems
9XDDN	7/09/91	6144	Access to Ada 9X information on DDN
9XNEWS	2/10/92	6144	Copy of the most recent Ada 9X Report to the Public
9XORDER	11/05/91	4096	How to order Ada 9X documents
ABSTRACT	12/20/91	20480	Abstracts of Ada-related articles
ACEC	8/15/91	61440	How to obtain the Ada Compiler Evaluation Capability (ACEC), DOD's compiler-performance test package
ACVC	4/29/91	40960	How to obtain a copy of the latest Ada Compiler Validation Capability (ACVC), the validation test suite
ADA-BIB	10/15/91	2048	How to obtain the AJPO'S Ada Bibliography, Volumes I, II, and III (1983–1986) and description
ADA-CALR	1/30/92	10240	List of upcoming conferences, symposia, and programs on Ada

ADA-DDN	8/06/91	6144	How to access ajpo.sei.cmu.edu, the AJPO host on the DDN
ADA-PROD	8/07/91	22528	List of articles and books on Ada costing, sizing, and productivity
ADA-RBBS	2/06/92	6144	How to access the AdaIC Bulletin Board at (703) 614-0215 or (301) 459-3865
ADA-USE	3/14/91	164839	Summary of the Ada Usage Database, which lists reported Ada projects from around the world
ADABOOKS	2/10/92	40960	Books relating to the Ada Programming Language
ADACPLUS	12/20/91	24576	Summary of Ada vs C++ Business Case Analysis Report
ADAIC	2/06/92	14336	A description of services offered by AdaIC
ADANET	3/15/91	4096	Text of AdaNet's Executive Summary describing its on-line services
ADATODAY	2/06/92	24576	On-line newsletter of current events, etc., relating to Ada
ADAYEST	2/04/92	36864	Items archived from Ada Today (ADATODAY.HLP)
AEO-SEO	1/14/92	4096	Current list of Software Executive Officials (formerly AEO)
AF-IMP89	7/18/89	29081	Text of 1/1/89 Air Force Ada Implementation Plan
AF-INT91	8/12/91	2048	Text of Air Force 1991 Interpretation of Congressional Mandate

AF-POL88	11/09/88	41809	Text of 11/9/88 Air Force policy on programming languages
AF-POL90	12/21/90	10868	Text of 8/7/90 Air Force policy on programming languages
AI-ADA	8/12/91	24576	Ada and Artificial Intelligence documents available from DTIC and NTIS
AJPO-891	10/28/91	6144	Article announcing that SPC's guide would be AJPO's suggested Ada style guide (with ordering info)
ARCHIVES	11/02/89	18341	Items archived from Ada Yesterday (ADAYEST.HLP) that are more than 1-year old
ARMIMP90	7/16/90	17928	Text of 7/16/90 Army Ada Implementation Plan
ASEETLIB	4/10/91	16446	Training-related materials in the ASEET Materials Library at the AdaIC
ATIP-F89	4/24/91	18432	Projects assisted by the Ada Technology Insertion Program in FY89
BENCHMRK	7/30/91	12288	How to obtain various benchmark performance test suites
BINDINGS	2/04/92	73728	Available Ada Bindings
CLAS-SEM	2/06/92	51200	Classes and seminars relating to the Ada language

CREASE	11/27/91	2048	How to obtain AJPO's Apr 88 Catalog of Resources for Education in Ada & Software Engineering (CREASE Ver 5.0)
CREASFOR			Ada Education Survey form for CREASE Ver. 6.0
DEF-MCCR	3/04/83	4795	Text of 3/4/83 DoD guidelines for acquiring computer resources (defines mission-critical computer resources)
DOCU-REF	12/05/91	20480	List of Ada-related documents available through DTIC/NTIS and information on how to obtain them
DOORS	7/09/91	6144	AdaIC Databases Available On-line Ada Products and Tools, and Ada Pragma Support
EMBDSYS	9/09/91	34816	Abstracts of documents and articles on Ada and embedded systems
FAA_ADA	11/07/89	6207	Text of 10/20/89 FAA Action Notice for mandating the use of Ada in acquisition and major modifications
GENINTRO	10/10/91	2048	Cover letter to accompany General Information Packet
GLOSSARY	8/11/90	47056	Ada-related terms and their meanings
GRAMM/	10/04/89	37569	"A LALR(1) Grammar for ANSI Ada" by Gerry Fisher and Phillipe Charles, 1983
HISTADA	11/26/91	26624	"The History of Ada"—March 1984 article by Robert DaCosta

IMPGUIDE	11/26/91	2048	How to obtain the Ada Compiler Validation Capability Implementers' Guide (1986)
ISO-STAT	11/26/91	10240	Background information on the ISO's acceptance of Ada as an international standard
LADY-LOV	11/25/91	10240	Article on life of Ada Lovelace by Carol L. James and Duncan E. Morrill with note on the naming of the Ada language
LRM	11/26/91	4096	How to obtain the Ada Language Reference Manual, ANSI/ MIL-STD-1815A 1983
MAIL_DDN	8/20/91	51200	A list of UNIX public-access sites that can be used to send E-mail to hosts on the DDN
MANDAT90	1/28/92	6144	Text of the Congressional Ada mandate—plus some background
MARIMP88	3/09/88	32563	Text of 3/9/88 Marine Corps Ada Implementation Plan
NATO-ADA	11/26/91	2048	Text of 1985 AJPO announcement of NATO's adoption of Ada as common HOL in military systems
NAVIPL91	11/26/91	20480	Interim Department of the Navy Policy on Ada, 24 June 1991
OODBIB	9/05/91	34816	List of articles and documents on Ada and object-oriented design
REALTIME	6/19/91	40960	List of publications on Ada real time

REPOSTRY	11/26/91	14336	How to obtain programs and tools from the Ada Software Repository on SIMTEL20
REUSCODE	2/06/92	16384	Sources of Ada source code, reusable components, and software repositories
REUSEPUB	9/16/91	24576	List of publications relating to the reuse of Ada source code
SERIALS	11/26/91	12288	List of serial publications that feature information on the Ada language and the Ada community
STYL-ORD	11/05/91	2048	Ordering information and order form to order version 2 of Ada Quality and Style
SUCCESS	10/17/91	34816	Reprint of article from *Military & Aerospace Electronics*
TNG-TAPE	11/25/91	20480	Description and ordering information for 19-tape series of Ada training videotapes
TRADEMRK	4/23/91	6144	Text of 1987 AJPO announcement that Ada trademark is replaced by certification mark
VAL-COMP	2/05/92	123280	List of currently validated Ada compilers
VAL-DOC	7/03/91	2048	How to obtain the Ada Compiler Validation Procedures
VAL-NOV	12/01/90	145846	List of validated Ada compilers as of Nov 90—kept for information purposes

VAL-PROC	9/19/90	55320	Text of the Ada Compiler Validation Procedures, Version 2.1, August 1990
VALCOVER	4/16/91	2048	Cover letter to accompany Validation packet
VALFACIL	12/04/91	2048	List of Ada Validation Facilities (AVFs) performing Ada Compiler Validation Capability tests
VSR-DOCU	7/03/91	24576	List of Validation Summary Reports (VSRs)—results from testing of compilers—and how to order info from DTIC/NTIS
WITHDRWN	8/05/91	8192	Tests that have been withdrawn from the validation test suite, ACVC 1.11
X-SURVEY	11/01/91	12288	X/Ada binding user questionnaire of the X/Ada Study Team at GHG Corporation

HELPFUL GOVERNMENT SOURCES MATRIX

SOURCE	PAGE	STDS	TRNG	REPS	B-BRDS	PUBS	CONF.	TOOLS
Ada AND C++	A-10					X		
Ada SLICES	A-11					X		
Ada SOFTWARE REPOSITORY	A-17			X				X
Ada TECH B-BOARD	A-16				X			X
Ada VALIDATION OFFICE	A-3	X						X
Ada9X PROJECT	A-2	X			X			
AdaIC	A-3	X	X	X	X	X	X	X
AdaSAGE	A-8		X		X			X
AJPO	A-2	X	X	X	X	X	X	X
ALS/N	A-7		X		X			X
ASSET	A-7		X			X	X	
CAIS	A-8	X	X					X
CRSS	A-20			X				
C2MUG	A-12					X		X
DACS	A-12					X		X
DON Ada REPRESENTATIVE	A-4	X	X	X		X	X	X
DON Ada USERS GROUP	A-23	X	X	X	X		X	X
LANGUAGE CONTROL FACILITY	A-13	X	X	X		X		X
NASIE POC	A-25		X					X
NATIONAL AUDIOVISUAL CENTER	A-9		X					
NATIONAL TECH INFO SERVICE	A-3					X		
NAVCOMTELCOM Ada REP	A-4		X	X	X			X
NAVY WIDE REUSE CENTER	A-21			X				X
RAPID	A-22			X				X
SEI	A-5		X			X	X	X
SPAWAR 224-1	A-4	X						
STARS	A-6			X		X	X	X
STINFO DOCS ORDER DESK	A-14	X				X		
U.S.ARMY ENG COLLEGE	A-10		X					
USMC Ada REP	A-5	X	X	X				

A.3 OTHER SOURCES

The commercial and non-profit organizations cited below are provided to help the DON Program Manager become knowledgeable about Ada-related issues. These sources are not endorsed by the DON. They are provided to augment the list of Government sources in Section A.1 and to help Program Managers become familiar with the wide array of available sources.

Other sources (e.g., organizations, training, publications, tools) to be considered for inclusion in future editions of the Ada Implementation Guide should be sent to the following address:

Commander
Space and Naval Warfare Systems Command
SPAWAR 2241 (CDR M. Romeo)
2451 Crystal Drive
Washington, D.C. 20363-5100

A.3.1 Training

Fastrak Training Inc.
Quarry Park Place
9175 Guilford Road
Suite 300
Columbia, MD 21046-1802
(301) 924-0050

Fastrak provides courses on topics such as Ada programming, evaluation of Ada code, and object-oriented requirements analysis and design.

Reifer Consultants Inc.
Marketing Manager
Reifer Consultants Inc.
2555 Hawthorne Boulevard, #208
Torrance, CA 90505-6825
(213) 373-8728

Reifer Consultants Inc., founded in August 1980, focuses primarily on consulting in Ada transition metrics, risk analysis, and cost estimating.

Telesoft
5959 Cornerstone Court West
San Diego, CA 92121
(619) 457-2700

Telesoft has been a major Ada compiler vendor for the last 9 years. Telesoft has developed a set of training courses tailored to the installation and use of its compiler technology.

Texel Company
Victoria Plaza, Building 4, #9
615 Hope Road
Eaton, NJ 07724
(201) 992-0232

Texel specializes in Ada education and training consulting, Independent Validation and Verification (IV&V), and application development.

A.3.2 Publications

AdaDATA Newsletter
International Resource Development, Inc.
P.O. Box 1716
New Canaan, CT 06840
(203) 966-2525

This monthly newsletter covers market trends and commercial developments in Ada software, services, and equipment. The cost of a subscription is $445 per year.

Ada Letters
Association for Computing Machinery, Inc.
1515 Broadway
New York, NY 10036
(212) 869-7440
Attn: Membership Services
acmhelp@acmvm.bitnet

This bimonthly publication for the ACM SIGAda, has been published since 1981. ($15 per year to ACM members; $44 per year for nonmembers. Annual ACM Membership dues—$75; Students—$22). The newsletter contains technical Ada articles as well as a calendar of Ada events.

Ada Monthly
Grebyn Corporation
P.O. Box 497
Vienna, VA 22183-0497
(703) 281-0497

This monthly newsletter, which has been published since 1987, contains information on upcoming events of interest to the Ada community, press releases, announcements by vendors of Ada-related products, and an up-to-date listing of validated Ada compilers. ($60 per year)

Ada Newsletter
Raytheon Equipment Division
Tim Boutin, Editor
MS 5-2-508
Sudbury, MA 01776
(508) 440-3607

This newsletter tracks developments in the Ada language through conference reports and provides vendor news articles and a listing of Ada events. There is no charge for this publication.

Ada Rendezvous
Texas Instruments Incorporated
David G. Struble
Software Engineering Department
MS 8503
P.O. Box 869305
Plano, TX 75086
(214) 575-5346

The Ada Rendezvous is a free biannual publication. Articles span multiple areas of interest including results of Ada compiler evaluations for embedded targets, review of Ada tools, and technical information contributed by Ada developers. Such articles provide guidance to application programmers on how to use Ada with specific hardware architectures and microprocessor designs. The Rendezvous also addresses evolving Government and DOD issues that affect existing and proposed contracts with Ada requirements.

Ada Strategies
Ralph E. Crafts, Editor and Publisher
Route 2, Box 713
Harpers Ferry, WV 25425
(304) 725-6542

This monthly newsletter covers Ada business strategies and contract-evaluation guidelines. It provides information on Ada policy and trends and on Congressional and funding issues as well as insight concerning current legislation. The annual cost is $100 for Government subscribers.

CAUWG Report
Alsys, Inc.
Lori Heyman
67 South Bedford Street
Burlington, MA 01803-5152
(617) 270-0030

This newsletter for members of ACM SIGAda's Commercial Ada Users Working Group (CAUWG) contains news and comments. It is available to the public at no charge.

FRAWG Newsletter
Martin Marietta Aerospace
MS L0420
P.O. Box 179
Denver, CO 80201
(303) 971-6731

This newsletter is a publication of the Front Range Ada Working Group (FRAWG). There is no charge for this publication.

Software Engineering Notes
Association of Computing Machinery, Inc.
1515 Broadway
New York City, NY 10036
(212) 869-7440
acmhelp@acmvm.bitnet

This quarterly is an informal publication of the ACM Special Interest Group on Software Engineering (SIGSOFT), which is concerned with the design and development of high-quality software. ($16 per year to ACM members, affiliate non-member subscription $38) (Annual ACM Membership dues—$75; Students—$22)

SPC Quarterly
SPC Building
2214 Rock Hill Road
Herndon, VA 22070
(703) 742-8877

This newsletter is a publication of the Software Productivity Consortium (SPC), a conglomerate of private companies that help one another reduce the time necessary for software development and the cost of building real-time software. The SPC provides computer-based software engineering methods, tools, and services to speed large-scale software development. SPC has a strong Ada orientation. Its methods and tools seek to make the Ada software developer more productive. Use of these methods and tools helps bridge the gap between well-established software engineering principles and the actual practice of programming in Ada. The newsletter is a source of information about companies that are making effective use of software engineering tools and Ada implementation. There is no charge for this publication.

A.3.3 Repositories

COSMIC, University of Georgia
382 East Broad Street
Athens, GA 30602
(404) 542-3265
FAX: (404) 542-4807

COSMIC distributes NASA-developed software including string, numerical, service, and linear algebra subprograms. Many are oriented to avionics applications. Source code is provided with purchase, and a free brochure is available.

EVB Software Engineering Inc
5303 Spectrum Drive
Frederick, MD 20910
1-800-877-1815
(301) 695-6060
FAX: (301) 695-7734

Generic Reusable Ada Components for Engineering (GRACE™) is a library of 275 Ada software components based on commonly used data structures such as strings, trees, and graphs. Each component includes complete design documentation, source code, and at least one test program. GRACE is completely portable. (Free samples available)

IWG Corp.
1940 Fifth Avenue
Suite 200
San Diego, CA 92101
(619) 531-0092

Proplink is an Ada program for analysis of communication link propagation paths from ELF to EHF.

MassTech, Inc.
3108 Hillsboro Road
Huntsville, AL 35805
(205) 539-8360
FAX: (205) 533-6730

Math pack contains over 320 Ada mathematical subprograms in 19 reusable generic Ada packages. It includes linear algebra, linear system solutions, integration, differential equations, eigensystems, interpolation, probability functions, Fourier transforms, and transcendental functions. Purchase includes source code, documentation, on-line help, and telephone support.

Rockwell International Corporation
Manager, Software Engineering Process Group
M/S 460-225
3200 East Renner Road
Richardson, TX 75081
(214) 705-0000

Rockwell International Corporation maintains a database of tools, software components, data, and the like that are accessible to all Rockwell software engineers.

Wizard Software
2171 South Parfet Court
Lakewood, CO 80227
(303) 986-2405

Booch components feature data types and tools for sorting, searching, and character matching. Each abstraction has multiple implementations and follows object-oriented design. Source code is provided. (Also marketed in Europe and Japan)

A.3.4 Conferences and Special Interest Groups

SIGAda

Mr. Mark Gerhardt
ESL, Inc. MS M507
495 Java Drive
Sunnyvale, CA 94088-3510
(408) 752-2459
(408) 738-2888 (Switchboard)

SIGAda is a professional society dedicated to the dissemination of information about all aspects of the Ada programming language, including standardization, implementation, usage, policy, management, and education. It sponsors meetings several times per year and also publishes a bimonthly newsletter, called *Ada Letters*. Originally known as AdaTEC, SIGAda was established under the auspices of ACM in 1981. In addition to the national SIGAda organization, there are approximately 50 chartered local SIGAda chapters. Most of these local chapters hold technical meetings on a monthly basis. The point of contact for each local chapter is published in *Ada Letters*. The Washington, D.C., chapter of SIGAda holds an annual symposium on Ada.

Tri-Ada Conference

Danieli & O'Keefe Associates, Inc.
75 Union Avenue
Sudbury, MA 10776
(508) 443-3330
1-800-833-7551 (in the United States and Canada only)
FAX +1 (508) 443-4715
(412) 268-777X

Tri-Ada, SIGAda's major annual conference and exposition, combines the availability of lectures about the technology and management of "what's happening" in the Ada community with in-depth presentations on project experience. The conference offers tutorials, birds-of-a-feather sessions, and the opportunity to see the Ada products and services available in the marketplace. In addition, information gathered in the coffee klatches and informal gatherings, which always occur at these meetings, is not

obtainable in any other way. Tri-Ada presents a unique opportunity to be immersed in the happenings in the world of Ada so that organizations can become or continue to be at the forefront of Ada understanding and use.

Washington Ada Symposium
Washington D.C. Chapter ACM
P.O. Box 6228
Washington, D.C. 20015
(301) 286-7631

At the Washington Ada Symposium (WAdaS), information is presented on software engineering dealing with commercial industry, Government, military, scientific, academia, weapons, and administration with regard to Ada.

A.3.5 Operational Support Development Tools

ObjectMaker
Mark V Systems Limited
16400 Ventura Boulevard, Suite 303
Encino, CA 91436
(818) 995-7671

ObjectMaker (formerly Adagen) is a Computer-Aided Software Engineering (CASE) tool that supports object-oriented diagramming methods for requirements analysis and top-level and detailed design. User-interface and diagram types are tailorable. Optional language modules automatically generate compilable code (C, C++, or Ada) from detailed design diagrams and support reverse engineering. The language module reverse engineering toolset takes legal code (C, C++, or Ada) back to multiple-level, nested, and detailed design-level diagrams. This is powerful for reuse and re-engineering as well as documenting as-built code and component libraries. Older methods supported include dataflow diagrams, real-time extensions, entity relationship, state transition, and structure charts. ObjectMaker is available on DEC, SUN, Apollo, Hewlett Packard, MIPS, and Data General Aviion workstations as well as Macintoshes and IBM PCs.

Appendix B
Supplementary Reading

This appendix lists publications, including Software Engineering Institute (SEI) reports on data rights, that are useful to Program Managers. Reports that have Defense Technical Information Center (DTIC) numbers are available from DTIC and the National Technical Information Service (NTIS) at the following addresses:

DTIC Defense Technical Information Center
ATTN: FDRA Cameron Station
Alexandria, VA 22304-6145

NTIS National Technical Information Service
U.S. Department of Commerce
Springfield, VA 22161

SEI reports that have a DTIC number (i.e., ADA followed by six digits) may be obtained directly from:

Software Engineering Institute
ATTN: Publications Requests
Carnegie-Mellon University
Pittsburgh, PA 15213-3890

SEI affiliates and Governmental organizations may order documents directly from SEI by submitting a written request, accompanied by a mailing label with the requestor's address, to the above address.

Data Rights Reports

Martin, A. and K. Deasy, *Seeking the Balance Between Government and Industry Interests in Software Acquisition. Volume I: A Basis for Reconciling DOD and Industry Needs for Rights in Software* (CMU/SEI-87-TR-13, ADA185742).

Martin, A. and K. Deasy, *The Effect of Software Support Needs on DOD Software Acquisition Policy: Part 1: A Framework for Analyzing Legal Issues* (CMU/SEI-87-TR-2, ADA178971).

Samuelson, P., *Understanding the Implications of Selling Rights in Software to the Defense Department: A Journey Through the Regulatory Maze* (SEI-86-TM-3, ADA175166).

Samuelson, P., *Comments on the Proposed Defense and Federal Acquisition Regulations* (SEI-86-TM-2, ADA175165).

Samuelson, P., *Adequate Planning for Acquiring Sufficient Documentation About and Rights in Software to Permit Organic or Competitive Maintenance* (SEI-86-TM-1, ADA175167).

Samuelson, P. and K. Deasy, *Intellectual Property Protection for Software* (SEI-CM-14-2.1/July 1989).

Samuelson, P. et al., *Proposal for a New "Rights in Software" Clause for Software Acquisitions by the Department of Defense* (CMU/SEI-86-TR-2, ADA132093).

Appendix C
The Initial Process

The following paragraphs on the Initial Process are from the Carnegie-Mellon University/Software Engineering Institute (CMU/SEI) publication, *Characterizing the Software Process: A Maturity Framework*, by Watts Humphrey (CMU/SEI-87-TR-11, June 1987.)

> The Initial Process could properly be called ad hoc or chaotic. Here, the organization typically operates without formalized procedures, cost estimates, and project plans. Tools are not well integrated with the process or uniformly applied. Change control is lax and there is little senior management exposure or understanding of the problems and issues. Since problems are often deferred or even forgotten rather than solved, software installation and maintenance often present serious problems.
>
> While organizations at the Initial Process may have formal procedures in place for project control, there is no management mechanism to assure that they are used. The best test is to observe how such an organization behaves in a crisis. If it abandons established procedures and reverts to merely coding and testing, it is likely to be at the Initial Process. In essence, if the process is appropriate, it must be used in a crisis and if it is not appropriate, it should not be used at all.
>
> One key reason why organizations behave in this chaotic fashion is that they have not gained sufficient experience to understand the consequences of such behavior. Since many effective software actions such as design and code reviews or test data analysis do not appear to directly support shipping the product, they seem expendable. It is much like driving an automobile. Few drivers with any experience will continue driving for very long when the engine warning light comes on, regardless of their rush. Similarly, most drivers starting on a new journey will, regardless of their hurry, pause to consult a map. They have learned the difference between speed and progress. In software, coding and testing seem like progress but they are often only wheel spinning. While they must be done, there is always the danger of going in the wrong direction. Without a sound plan and a thoughtful analysis of the problems, there is no way to know.
>
> Organizations at the Initial Process can advance to the Repeatable Process by instituting basic project controls. The most important are:

1. **Project Management.** The fundamental role of a project management system is to insure effective control of commitments. This requires adequate preparation, clear responsibility, a public declaration, and a dedication to performance. For software, this starts with an understanding of the magnitude of the job to be done. In any but the simplest projects, a plan must then be developed to determine the best schedule which can be met and the anticipated resources required. In the absence of such an orderly plan, no commitment can be better than an educated guess.

2. **Management Oversight.** A suitable disciplined software development organization must have corporate oversight. This includes review and approval of all major development plans prior to their official commitment. A quarterly review is also conducted of facility-wide process compliance, field quality performance, schedule tracking, cost trends, computing service, and quality and productivity goals by project. The lack of such reviews typically results in uneven and generally inadequate implementation of the process as well as frequent overcommitments and cost surprises.

3. **Product Assurance.** A product assurance group is charged with assuring management that the software development work is actually done the way it is supposed to be done. To be effective, the assurance organization must have an independent reporting line to senior management and sufficient resources to monitor performance of all key planning, implementation, and verification activities. This generally requires an organization which is between 5% and 10% of the size of the development organization.

4. **Change Control.** Control of changes in software development is fundamental to business and financial control as well as to technical stability. To develop quality software on a predictable schedule, the requirements must be established and maintained with reasonable stability throughout the development cycle. Changes will have to be made, but they must be managed and introduced in an orderly way. While occasional changes are essential, historical evidence demonstrates that the vast bulk of changes can be deferred and phased in at a subsequent point. If change is not controlled, orderly testing is impossible and no quality plan can be effective.

Appendix D
Source Line Of Code Metrics Definition

This definition is useful for measuring the size of software applications in Ada and other languages. This definition provides a consistent metric for all DON activities.

The Source Line of Code (SLOC) elements are as follows:

- TLOC—Total Lines of Code
- SCL—Stand-alone Comment Lines
- ECLS—Embedded Comment Lines in Source
- ECLD—Embedded Comment Lines in Data
- SLOCWC—Source Lines of Code Without Comments
- TCL—Total Comment Lines
- DE1—Data Elements in the Source

The elements are related by the following formulas:

TLOC = SLOC + SCL
SLOC = ECLS + ECLD + SLOCWC
TCL = SCL + ECLS + ECLD

In addition, we track DEl, which may number more than one per SLOC.

The SLOC elements are described below.

TLOC: The TLOC is defined as the total SLOC plus the total SCL.

SLOC: The SLOC is defined as the lines of source code that are generated by the software engineer and that can be classified as statements. Each statement is counted as one SLOC. Data lines, code lines, and code lines with comments all would be included in this value. Lines with only comments are counted not in SLOC, but in SCL.

SCL: The SCL is defined as the lines of code that are solely comments, as separated by carriage returns, and are not "attached" to a construct or action line or statement. SCL does not produce object code for the target. SCL includes, for example, blank lines, pseudo-op pairs in an assembler, and decorative format lines. The point here is to understand how much documentation is included in the source code because SCL also has a value in the accounting of the software development.

ECLS: The ECLS is defined as comments that are "attached" to existing statements. ECLS lines produce object code for the target. An embedded comment is any comment that is included with a single statement; the comment may be at the end of the line or within the statement itself. The point of ECLS and ECLD is to determine how much documentation is included and/or required in the source for annotation and/or explanation.

ECLD: The ECLD is defined as comments that are "attached" to existing data statements. ECLDs produce object code for the target. An embedded comment is any comment that is included with a single statement; the comment may be at the end of the line or within the statement itself.

SLOCWC: The SLOCWC is defined as the total SLOC minus the total ECLS and ECLD.

TCL: The TCL is defined as the total SCL plus the total ECLS and ECLD.

DEl: The DE1 is defined as the individual data unit definitions in the source that are defined with preset values prior to run time and that are a part of the program size. More than one data element per SLOC may exist; each data element should count as one DEl and may be a table, variable, array, etc.

Note: These definitions evolved from *What is a line of Code?* by Barry Corson and Luke Campbell. The paper includes an abstract, background, motivation, and examples. The definitions are applicable for most languages used by the Department of Defense (DOD) and hence provide a foundation to compare software developments across a variety of languages. Automatic SLOC counters are under development. Examples provided in the paper include the following languages: Ada, Assembly, COBOL, and FORTRAN. The paper can be obtained from:

Mr. Luke Campbell
NATC Code SY30
Building 2035
Patuxent River, MD 20670

Phone: (301) 862-7601
FAX: (301) 862-7607
email: lcampbell@paxrv-nes.navy.mil

Appendix E
Example of Metric Wording for Use in a Contractual Document

Software Management Metrics Requirements

The contractor shall include graphs of Software Management Metrics (SMM) in the Software Developmental Status Report (SDSR). The abscissa of each graph shall contain the calendar months of the program and shall depict the times of System Requirements Review (SRR), System Design Review (SDR), Preliminary Design Review (PDR), and Critical Design Review (CDR). Should SMM data change as SDSRs are presented, the contractor shall always show the original estimate together with the current estimate and indicate the changes since the last estimate.

SMM data shall be depicted for all software, regardless of whether the prime contractor and/or subcontractors (if any) are involved in the development and whether the software is newly developed, existing, or reused.

The information shown shall be as specified below.

1. Software Size

The contractor shall use an automated tool to estimate the size of the software that needs to be developed and shall report this estimated size using the definitions from Attachment 1

During actual software development, the contractor shall report the Source Lines of Code (SLOC) elements in accordance with the information provided in the attachment to this appendix. SLOC metrics shall be provided for each Configuration Item (CI) for each programming language used. The contractor shall utilize an automated Code Counting Program (CCP) to provide the SLOC metrics results.

On a single graph, the contractor shall show the current values of total, new, reused, and modified SLOC counts.

2. Design Complexity

As software is developed, for each programming language used, the contractor shall use the appropriate static analyzer of the Verilog Logiscope tool to show flow graphs, call graphs, and Kiviat diagrams for each CI.

3. Software Personnel

The contractor shall record the number of engineering and management personnel supporting software development in experience categories of 1 through 3 years, 4 through 8 years, and 9 or more years. Software system planning, requirements definition, design, coding, test, documentation, configuration management, and quality assurance personnel shall be included.

For each contractor development organization, provide graphs showing planned and actual personnel in the various experience categories.

4. Software Volatility

The contractor shall provide three different graphs showing software volatility on the ordinate. One graph shall contain the total number of "shall" statements (requirements) in the Software Requirements Specifications (SRSs) and the cumulative number of requirements changes (including additions, deletions, and modifications). A second graph shall contain new and cumulative Software Requirements Changes (SRCs), which are the number of unresolved requirement and/or design issues. A third graph shall depict Software Action Items (SAIs) that have been open from 1 to 45 days, 46 to 90 days, or over 90 days.

5. Computer Software Unit Development Progress

The contractor shall graph the progress made in Computer Software Unit (CSU) development against initial plans. This progress shall be reported to show monthly planned versus actual progress of the number of CSUs designed, tested, and integrated.

6. Testing Progress

The contractor shall record and graph the progress of CI and system testing against initial plans and the degree to which the software is meeting requirements. One graph shall depict the number of CI tests planned and passed, together with the number of system tests planned and passed. A second graph shall depict the number of new Software Problem Reports (SPRs) per month and the SPR density, which is

the cumulative number of SPRs per 1000 SLOC. A third graph shall depict the cumulative number of open SPRs and the number of SPRs that have been open from 1 through 45 days, 46 through 90 days, or over 90 days.

7. Build Release Metric

The contractor shall present a graph that contains each build, or release, of the software, showing the number of originally planned versus currently planned CSUs for each release.

8. Computer Resource Utilization

The contractor shall record the utilization of each target computer resource, including memory (all types), Input/Output (I/O) channels, I/O bandwidth, processor throughput under various extreme system loads, expected "normal" system load (including I/O), and memory use during processing times. Utilization metrics shall be proposed by the contractor and approved by the Government. The data shall show the planned versus actual utilization for each target computer resource.

In addition, the contractor shall report on availability and use of host development station(s) to show planned versus actual usage.

Attachment
SLOC Metrics Definitions

This attachment details the SLOC elements and their definitions. The elements are divided into Source Elements and Target Elements. To show how the proposed elements "add up," the following formulas have been provided.

Source Elements

(Total created lines of code) = (SLOC) + (SCL)
where SLOC is Source Line of Code and SCL is Stand-alone Comment Line.

(SLOC)= (ECLS) + (ECLD) + (Lines with no comments)
where ECLS is Embedded Comment Lines in Source and ECLD is Embedded Comment Lines in Data.

In addition, we track Data Elements (DEl), which may number more than one per SLOC.

Target Elements

(OFPS) = (Data) + (DBuf) + (all other code generated by source program)

where OFPS is Operational Flight Program Size in words and DBuf is Target Code Size of Buffer Data in words.

Finally, the hope is that the Target Capacity (TC) is somewhat larger than OFPS. This is not always true, however, as some platforms have overlays.

SOURCE ELEMENTS

(1) Number of SLOC—the lines of source code that are generated by the software engineer and that can be classified as statements. Each statement is counted as 1 SLOC. Data lines, code lines, and code lines with comments would all be included in this value. Lines with only comments are not counted in SLOC but in SCLs (see item 3 below).

(2) Number of Compiled/Assembled Lines of Code (CLOC)—lines of code in the final listing that are created by the compiler or assembler. This count would include any expanding constructs such as assembler or compiler directives (e.g., loops or

repeat statements). In many cases, the compiler or assembler numbers the lines on the listing, and the last value on the last page of the listing can represent CLOC.

(3) Number of Stand-alone Comment Lines (SCL)—lines of code that are solely comments, as separated by carriage returns, and that are not "attached" to a construct or action line or statement. SCL does not produce object code for the target. SCL includes, for example, blank lines, pseudo-op pairs in an assembler, and decorative format lines. The point here is to understand how much documentation is included in the source code because SCL also has a value in the accounting of the software development.

(4) Number of Embedded Comment Lines in Source (ECLS)—comments that are "attached" to existing non-data statements. ECLS lines produce object code for the target. An embedded comment is any comment that is included with a single statement; the comment may be at the end of the line or within the statement itself. The point of this element (and ECLD) is to determine how much documentation is included and/or required in the source for annotation and/or explanation.

(5) Number of Embedded Comment Lines in Data (ECLD)—comments that are "attached" to existing data statements. ECLDs produce object code for the target. An embedded comment is any comment that is included with a single statement; the comment may be at the end of the line or within the statement itself.

(6) Number of Data Elements in the Source (DEl)—individual data unit definitions in the source that are defined with preset values prior to run time and that are a part of the target's object code. There may be more than one data element per SLOC; each data element should count as one DEl. DEls may be tables, variable arrays, etc.

TARGET ELEMENTS

(7) Target Code Size of Data in words (Data)—the total number of target words needed for the storage of DEl. **Note that word size is determined by the target.** Data=(#bits/type)/(#bits/word), where type is a character, integer, float, double, etc.

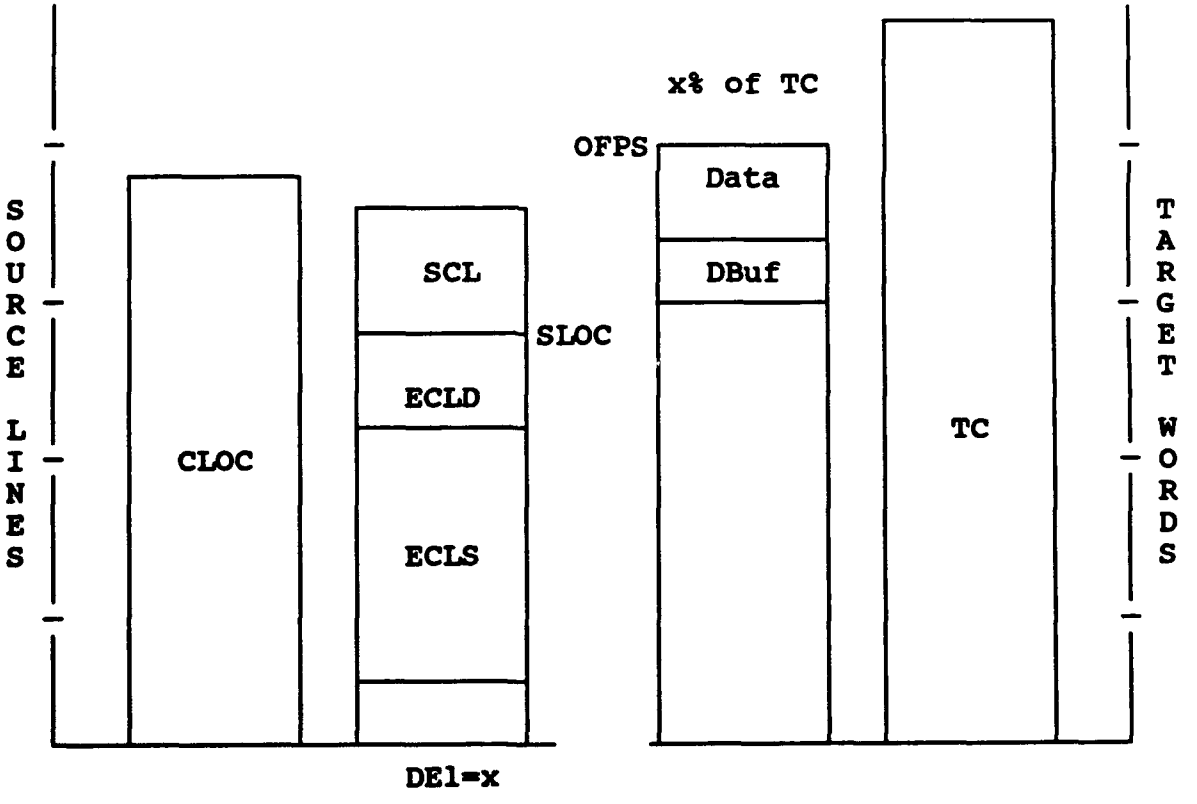
(8) Target Code Size of Buffer Data in words (DBuf)—the total number of target computer words NOT preset to a value prior to run time (e.g., buffers used for I/O and/or data gathered in flight).

(9) Target Code Size of Operational Flight Program in words (OFPS)—the total number of words of machine code for the entire OFP, including all types of code, data, and buffers.

(10) Target Memory Capacity in words (TC)—indication of the target's memory capacity in words. **(Note that if the OFP is stored in different types of memory, indicate the memory types and the percentage of data and code stored in each.)**

SLOC metrics shall be reported using an automated Code Counting Program (CCP) in the following format:

SLOC Data for SCI: _____ Date: _____
Language:_____ Host: _____ Target: _____



Appendix F
Software Tool Descriptions

Editor

The editor is a tool for text manipulation. When computers were in their infancy, source program text was entered by paper tape or punched card. Today, editors are sophisticated interactive screen/window-management tools. Modern editors are used not only for creating or modifying source text but also for viewing or modifying files produced by other tools.

An editor is used primarily to create or modify source program text. The product of the editor is a file that contains the source program statements. Because these program files always will have to be compiled, the advantage was recognized of having a language-specific editor, a tool with some built-in specific language requirements. These editors simplify the entering of program text and sometimes perform on-line error checking.

In its most elementary form, a language-specific editor may have special options to assist in formatting the source text. An example for FORTRAN would be automatically starting a line in column 7 whenever the first character was alphabetic, thus preventing text from being placed in the field reserved for line numbers.

Another form of a language-specific editor is the "syntax-directed editor," which is tightly linked to the programming language. Most modern languages require opening and closing statements for structured programming constructs. A syntax-directed editor can provide templates for these constructs. For example, the following template could be rapidly placed on the screen after typing "if":

```
if <condition>
then
....;
....;
....;
else
....;
....;
endif;
```

In addition, the syntax-directed editor can check the structure of the source text for compliance with the rules of the language. Thus, the efficiency of the edit-compilation process is improved because many programming errors are eliminated before compilation.

Compiler

A compiler is a program that translates a High Order Language (HOL) source program into its relocatable code equivalent. The term "host" refers to the computer that translates the source program, and the term "target" refers to the computer that will execute the compiled code. The term "cross-compiler" refers to the case where the target computer is different from the host computer. In many DON applications, a source program is cross-compiled on a host computer (generally a commercial machine) for a militarized target computer that is embedded in a system.

Compilers are usually multiple pass programs that may process the source program or some intermediate form several times before completion. The output of the earlier stages is referred to as intermediate code. In some host computer systems, the intermediate code is used by other tools.

Compilers for real-time applications must produce code to fit in limited storage space. In addition, the execution speed on the target computer must allow all of the required functions to be computed in the assigned time.

Assembler

An assembler is a program that translates an Assembly language source program into relocatable code. Note that, usually, a one-to-one correspondence exists between an Assembly language source statement and a machine instruction. Assemblers allow the programmer to use relative addressing and then specify a starting location rather than having to specify each address in absolute terms. Most assemblers also allow the use of labels and other defined values and locations.

The Assembly language has been used frequently in Mission-Critical Computer Resources (MCCR) applications because it allows the programmer to optimize storage space and execution time. However, Assembly programs are difficult to test and expensive to maintain. Today, the use of Assembly is generally restricted to routines with exceptionally high performance requirements and to hardware diagnostic software.

Linker

Source program modules, whether in an assembly language or a HOL, usually are translated separately into modules of relocatable code. Once translated, the modules

must be linked together before execution. A linker is a program that creates a load module from one or more independently translated modules by resolving the cross-references among the modules.

Relocating Loader

Relocatable code contains relative addresses of machine instructions and data. This defers the assignment of absolute addresses until the program is ready for execution and allows the flexibility of placing the program in any contiguous block of storage. The linker creates a load module that leaves all addresses in relative form although it has resolved the cross-references between modules. The relocating loader is a program that executes on the host computer and translates the relative addresses into the absolute addresses; its output is an execution module. A bootstrap loader executes on the target computer and copies the execution module into its storage.

Run-Time Executive

The Run-Time Executive resides on the target machine and provides a variety of services for application programs. Typical Run-Time Executive functions are dynamic storage management, exception processing, input and output, and task scheduling. Because this executive is used for all application programs, it should be small and fast to minimize the overhead. The Run-Time Executive usually is modularized according to the particular services it provides and is automatically configured when the execution module is created. Thus, if an application program does not need a particular service, that module is automatically omitted from the Run-Time Executive.

Simulator/Emulator

When producing code for a target computer that is different from the host, the problem of how to test the code must be considered. Testing on the target computer is usually difficult because the computer may be still under development; it may be being integrated with other embedded subsystems; or the number of target machines may be insufficient to support all of the programmers. Moreover, most embedded target computers have poor tools to support testing.

One solution to this problem is to build a software simulator or emulator of the target computer that executes on the host computer. A software emulator accepts the same data, executes the same instructions, and achieves the same results as the target machine. A simulator imitates selected features of the target computer but is not required to achieve identical results. The best tool is a target computer emulator that can operate in either batch or interactive mode. The execution speed of an emulator may be significantly slower than that of the target computer. However, the emulator has many advantages. For example, the emulator can be time shared and used by everyone on the host computer; because the emulator is on

the host computer, it is easy to generate test data, load the module and test data into the emulator, and monitor the test while in progress; and long tests can be run in batch mode during off-peak hours.

In-Circuit Emulator
An in-circuit emulator (ICE) provides the user with a means of executing a software program located in external Random Access Memory (RAM) rather than internal Read Only Memory (ROM) or Erasable Programmable Read Only Memory (EPROM.) This allows programs being debugged to be modified easily and quickly during the testing cycle. When connected to the prototype system through the microprocessor socket, an ICE can emulate, test, and trace the prototype system operation. The internal state of the microprocessor, including RAM, accumulator, internal working registers, and stack and status registers, can be observed and modified. Some ICEs allow the recording of data bus operations. This feature allows the engineer to capture N events before or after a failure or predefined occurrence.

Symbolic Debugger
A symbolic debugger allows a programmer to test a module by controlling the program execution on a target computer emulator or the target computer itself. With the symbolic debugger, the programmer can address the variables by using their source program symbols or names. The facilities generally provided include stop execution at selected locations, single step in increments of source statements, watch the value of specified variables, trace execution, examine the contents of variables, evaluate expressions, display the current sequence of routine calls, display the source corresponding to any part of the program, execute debug command procedures at break points, and call procedures that are program parts.

Pretty Printer
A pretty printer is a program that automatically applies standard rules for formatting program source code. It will accept an input file and format the text to match a style guide. For example, a pretty printer for a block-structured language will produce a listing where the indentation level of each block shows its nesting level. A pretty printer helps the programmer read and comprehend the program. After extensive program modifications, for example, it helps eliminate confusion about the program structure and nesting levels.

Host-to-Target Exporter
If the target machine is different from the host machine, it is necessary to have a tool to transmit the execution module from the host to the target. Standard communications software and hardware may be used, but these are rarely available for embedded machines.

It is desirable to have a flexible, high bandwidth communications link between the host and target. If the link has a "pass-through" capability, then an interactive user of the host computer can run tests on the embedded computer from the same terminal. High bandwidth is desirable because large volumes of data must be exchanged between the host and target; for example, diagnostic software is typically sent to the target to test the target hardware, and test data and test results also are exchanged. A high bandwidth communications link will reduce the time it takes to do these tasks and allow more time for testing.

Appendix G
Ada Bindings and Secondary Standards

Currently, a large number of Ada bindings are available. "Available Ada Bindings" can be obtained from the Ada Information Clearinghouse (see Appendix A, Helpful Sources). Appendix H shows the status of Ada bindings.

G.1 PORTABLE OPERATING SYSTEM INTERFACE FOR UNIX

NAME:	POSIX—Portable Operating System Interface for UNIX
SPONSOR:	Institute of Electrical and Electronics Engineers (IEEE) and International Standards Organization (ISO)
STATUS:	See Figure G-1
STATUS OF ADA BINDING:	See Figure G-1
DOD APPLICATIONS:	Virtually all software applications

IMPORTANCE TO DOD: POSIX defines a collection of system services that will be widely supported by computer vendors and third-party software developers. The fundamental goal of POSIX standards is to support applications program portability by defining interface standards.

DISCUSSION: POSIX is the name for the IEEE standardization efforts to develop portable application interfaces to operating systems. The effort is coordinated with UniForum, X/Open, National Institute of Standards and Technology (NIST), and other organizations. The resulting IEEE standards follow processes that permit simultaneous standardization as ISO standards. The IEEE standards typically become Federal Information Processing Standards (FIPS). Each POSIX standard is required to include test assertions and test methods that can be used to test implementation conformance.

The POSIX interfaces are based on existing UNIX interfaces, but it is important to note that many non-UNIX systems, such as Digital Equipment Corporation's VMS, are planning on POSIX compliance. The effort is broken into different areas that cover the range of interfaces used to access operating systems:

- Basic System Services—process management, file handling, terminal communications, and basic protection mechanisms
- Real-Time Services—real-time scheduling, synchronization, and process control
- Security Services—discretionary and mandatory security interfaces and definitions
- User Command Interface—command syntax, user services, and utilities
- User Graphical Interface—basic windows, toolkits, and user interface management systems
- Network Services—transnetwork transparent file access, process-to-process communication, directory name services, and generic data exchange
- Mail Services—user name resolution and mail delivery services
- System Administration—system control, integrity monitoring, failure recovery, and activity monitoring

Figure G-1 shows the status for each of the above POSIX standardization areas. Note that Ada binding and implementation efforts have not even begun for many of these areas.

While the POSIX effort covers a wide range of operating system interfaces, several notable areas frequently are considered parts of operating systems but are not considered to be in the POSIX domain. Specifically, Database Management Systems (DBMSs), graphics other than user interface (e.g., Graphical Kernel System [GKS], Programmer's Hierarchical Interactive Graphics System [PHIGS]), and knowledge-based systems are excluded from POSIX.

POSIX standardization efforts also include standard profiles that bring together a set of standards and narrow the options under those standards. Currently, POSIX defines profiles for transaction processing, including supercomputing and multi- and real-time processing. The efforts in these areas began only recently. These profile areas each may result in more than one profile, depending on how the application domains shake out.

Figure G-1. POSIX Status

POSIX AREA	STANDARD EFFORT	WIDE-USE INTERFACE	WIDE-USE ADA BINDING	ADA BINDING IMPLEMENTATION
Basic System Services	Done: ISO 9945-1 IEEE 100301 FIPS 151-1	All Major Vendors	In Ballot, Not Yet Stable	Trial Implementation Done To Earlier Revision
Real-Time Services	Stable, in Ballot	No	First Cut Done	No Work Yet
Security Services	Stable, Going to Ballot	No	No Work Yet	No Work Yet
Program Graphical Interface	No Work Yet	Some	Some	Some
Network Services	1 Year of Work, 1 Year From Ballot	No, but TCP/IP, X.500	No Work Yet	No Work Yet
Mail Services	6 Months of Work, 1 Year From Ballot	No, but X.400	No Work Yet	No Work Yet
User Commands	Stable, in Final Ballot	All Major Vendors	Not Applicable	Not Applicable
System Administration	2 Years of Work, 6 Months From Ballot	All Major Vendors	Not Applicable	Not Applicable

G.2 STRUCTURED QUERY LANGUAGE

NAME: Database Management Services:

1. Database Management System Component—Database Language Structured Query Language (SQL), American National Standards Institute (ANSI) X3.135.1-1989 (ISO 9075, FIPS 127); Embedded SQL, ANSI X3.168-1989

2. Data Dictionary/Directory Component—Information Resource Dictionary System (IRDS) ANSI X3.138-1989 (ISO DP 10027.N2642 (1988) IRDS Framework; ISO DP 8800 N2132 (1988) IRDS Services Interfaces; FIPS 156)

3. Distributed Data Component—Remote Database Access (RDA) draft specification, ANSI (ISO DP 9579, N2971)

SPONSORS:

1. Database Management System Component—ISO/International Electro-Technical Committee (IEC) Joint Technical Committee 1 (JTC1)

2. Data Dictionary/Directory Component—ANSI X3H4 Database Group and ISO/IEC JTC1

3. Distributed Data Component—European Computer Manufacturing Association (ECMA) Technical Committee on Database Standards TC22 and ISO/IEC JTC1

STATUS OF BASE STANDARD

1. These standards, developed between 1982 and 1989 by the ANSI X3H2 Database Committee, specify data access capabilities based on SQL and the relational database model. The X3.135 standard specifies data definitions, data manipulation, access control, and limited integrity constraints. An emerging enhanced specification under active development in both ANSI and ISO will include substantial additional features for schema manipulation, Dynamic SQL, exception handling, enhanced integrity constraints, transaction management, and data administration.

2. The ANSI standard (ANSI X3.138-1988) and the FIPS are the same document. ISO is working on an IRDS specification. This specification includes user interfaces only. Data dictionary/directory services consist of utilities and systems necessary to catalog, document, manage, and use metadata (information about data).

3. Distributed Data Component—The ISO/IEC RDA specification currently is undergoing draft balloting in ISO/IEC JTC1. The specification is in two parts: Part 1—Generic RDA, and Part 2—SQL Specialization. Final adoption is expected in 1992. Vendor consortia, such as SQL-Access and X/Open, are working on prototypes to demonstrate interoperability among different SQL servers.

STATUS OF ADA BINDING

1. Database Management System Component—X3.135 deals with SQL independently of programming languages. X3.168 binds or embeds SQL within the programming languages Ada, COBOL, FORTRAN, Pascal, PL/1, and C. DOD does not endorse this binding method. A module language binding has

been jointly developed by the AJPO and SEI. Entitled the SQL Ada Module Extension (SAME), this specification is approved by DOD but is not a formally endorsed standard. SEI is coordinating with ISO to establish the SAME Description Language (SAMeDL) as an international standard. The document is now at Working Group 9 in committee draft.

2. Data Dictionary/Directory Component—No current effort exists to develop an Ada language binding to the IRDS, and there are no commercial implementations of such a binding.

3. Distributed Data Component—No current effort exists to develop an Ada language binding to the RDA specification. All current Ada bindings are proprietary.

DOD APPLICATIONS

1. Database Management System Component—This component consists of definitions, management, and query of structured data stored in a relational database management system.

2. Data Dictionary/Directory Component—Data dictionary/directory services consist of utilities and systems necessary to catalog, document, manage, and use metadata.

3. Distributed Data Component—This component is used to establish a remote connection between a database client, acting on behalf of an application program, and a database server, interfacing to a process that controls data transfers to and from a database. The goal is to promote interconnection of database applications among heterogeneous environments, with emphasis on an SQL server interface.

IMPORTANCE TO DOD: Database management services include the data dictionary/directory component for accessing and modifying metadata, the DBMS component for accessing and modifying structured data, and the distributed data component for accessing and modifying data from a remote database. The relational data model has been adopted as the standard data model for use in developing applications. DBMSs provide a robust environment of database services and functions. The system designs of software applications are not limited to stand-alone computer systems. The term system now refers to a terminal (dumb or smart) connected by some means, either direct or through a network, to a processor. SQL provides a standard language for designing databases and manipulating the data in

them across multiple applications. Information about these data must be standardized and the method of accessing them common if DOD is to achieve interoperability and portability of applications across a large range of systems.

G.3 XWINDOWS

NAME:	XWindows
DEVELOPER:	MIT X Consortium
BASE STANDARD STATUS:	Within POSIX Standards Working Group
CURRENT RELEASE:	X11R4
ADA:	No Standard or Stable Ada Interface

DISCUSSION: Demand and use of graphical window systems that provide interfaces to users currently are increasing. This increase is due primarily to the release of the MIT X Consortium XWindows System. Within DOD, programs often require XWindows and different toolkits and extensions such as "the contractor shall provide XWindows, PHIGS, and the MOTIF toolkit." This requirement often creates despair due to the different levels of X and its compatibility with other graphics applications. In addition, at the higher levels of XWindows, such as the toolkit level, various competing de facto standards make the progress complex. Each level is described below, and the status of standardization and implementation at each level is outlined.

- Xlib—communicates with the X System Server through X Protocol; manipulates X primitives; provides the calls back to the XServer; provides mechanisms for efficiency, such as caching; and hides the network protocol from the programmer.
- Xt Intrinsics—sets of functions for basic user interface abstractions. It is layered above the Xlib to provide the user easier mechanisms for writing X applications.
- Toolkits—sets of basic objects (e.g., menus and buttons) built from a combination of Xlib and Xt Intrinsics calls. Different widget sets have individual "look and feel." Sample widget sets are Athena, HP, OSF/MOTIF, and OLIT.

- Virtual Toolkits—common high-level interface between application programs and native window systems. Virtual toolkits allow portability of window applications across a variety of different widget sets (e.g., MOTIF, MacTools) and across different platforms (e.g., SUN, Mac, DEC).

- User Interface Management Systems (UIMS)—systems that allow application developers to rapidly prototype user interfaces. UIMS often provides a user with the capability to quickly construct menus, dialogue boxes, and the like and to generate code for these applications.

G.4 GOVERNMENT OPEN SYSTEMS INTERCONNECTION PROFILE

NAME Network Services

1. GOSIP, FIPS 146
   - Message Handling Service (MHS), X.400 Series, ISO 10021
   - File Transfer, Access, and Management (FTAM), ISO 857, 8613, 10026, 8650, 8652, 8653, 9735, 9594
   - Virtual Terminal (VT), ISO draft
   - X.500 Directory Services
2. Transparent File Access (TFA)—IEEE P1003.8, Draft 3
3. Distributed Computing Services—Open Software Foundation (OSF)/1 Network Computing Services (NCS) Remote Process Communication (RPC)—OSF

SPONSORS

1. GOSIP, Version 1—ISO and International Consultative Committee for Telegraph and Telephone (CCITT)
2. TFA—IEEE
3. Distributed Computing Services—OSF

STATUS OF BASE STANDARD:

1. GOSIP—GOSIP is essentially a family of protocols and representations. GOSIP Version 1.0 provides a complete transparent, end-to-end data communications capability based on Open Systems Interconnection (OSI) Transport Class 4 (TP 4) and Connectionless Network Protocol (CLNP). Version 1.0 provides electronic mail and file transfer access and management applications. It operates over a variety of local and wide-area network technologies. GOSIP Version 2.0 will add remote logon and office document interchange applications, will provide a new network addressing structure to support dynamic routing, will include a provision to operate over the Integrated Services Digital Network (ISDN), and will allow an optional connectionless transport service to support transparent file access and other applications. Later versions of GOSIP will include substantial added functionality such as distributed directory services, transaction processing, remote database access, dynamic routing, network management, and network security. Key features may vary with specific combinations of vendor products and users. Layer 7 of the OSI Reference Model generally includes components that are considered Application Service Interfaces. This interface is used by application programmers to access high-level network services such as file transfer and mail.

2. TFA—TFA includes capabilities for managing files and transmitting data through heterogeneous electronic transmission networks in a manner that is transparent to the user (i.e., does not require knowledge of file location or of certain access requirements). Many functions of TFA are widely available in existing vendor implementations, but some functions rely on the underlying protocols. The eventual TFA specification should overcome this limitation.

3. Distributed computing services—These services include specifications for remote process communications and distributed real-time support in heterogenous networks. Distributed access services include functional support for submitting, starting, and stopping processes among processors in a network. No specifications exist that define a complete set of functions necessary to provide remote procedure communications for all types of application platforms (i.e., the language-independent representation of remote procedure calls).

STATUS OF ADA BINDINGS: No efforts are ongoing to develop an Ada language binding to the high-level network services provided by GOSIP protocols.

DOD APPLICATIONS: GOSIP is applicable to all data communications environments where multivendor computing is anticipated. GOSIP is based on OSI standards, the worldwide consensus standards for multivendor data communications.

IMPORTANCE TO DOD: GOSIP will allow interoperability of systems used within DOD and its international allies. Network services are the standard services needed to transport data between any two service access points within connected OSI-compliant systems and to manage the data communications infrastructure.

G.5 GRAPHIC STANDARDS

NAME:	GKS—Graphical Kernel System PHIGS—Programmer's Hierarchical Interactive Graphics System
SPONSORS:	ANSI and ISO
STATUS:	GKS is complete, but a major revision is in progress. PHIGS is complete.
STATUS OF ADA BINDING:	Ada/GKS binding is provided for the upper level, but not the lower levels, of GKS. Ada/PHIGS binding has been published by ISO and ANSI.
DOD APPLICATIONS:	Graphical displays, cartography, user interfaces
IMPORTANCE TO DOD:	Transportability of graphics software and compatibility of software with a variety of input/output devices

DISCUSSION: The GKS is an ANSI Standard (X3.124-1985) and an ISO Standard (ISO 7942) that defines a set of language- and device-independent types and operations that facilitate the programming of two-dimensional color graphics applications. To be used directly in a program, the GKS entities must be mapped to a particular programming language. Such a mapping, or binding as it is generally called, has been carried out with FORTRAN, Pascal, C, and Ada. The Ada Language Binding to GKS is also an ANSI Standard (X3.124-3) and ISO Standard (ISO 8651-3).

GKS includes the following concepts:

- Graphical output primitives—The basic building blocks under GKS are four main primitives. Polyline draws a sequence of connected line segments. Polymarker marks a sequence of points with a designated symbol. Fill Area displays a closed area (polygon). Text displays a character string and is supported by attributes such as character height, character spacing, and character up vector.

- Coordinate systems—GKS distinguishes between user and device coordinates and supports three different coordinate systems. The World Coordinates (WCs) system is a device-independent system specified in user coordinate space by the application programmer. The Normalized Device Coordinate (NDC) system is an intermediate system used by GKS that maps the WC into a virtual space on the x and y axes from 0 to 1. The Device Coordinate (DC) system is device dependent, and coordinates are expressed in terms of the device being used.

- Segments—This group of graphical primitives is identified by a unique name. Segmentation allows the collection of output primitives to be manipulated as a unit.

- Logical input devices—A logical input device is an abstraction of one or more physical input devices. Each logical input device can be operated in three modes: Request, Sample, and Event.

- Workstation—This is a GKS abstraction of graphical devices. The graphical workstation provides the logical interface through which the application program controls the physical devices.

GKS also provides an interface to a system for filing a graphical image for long-term storage and exchange of graphical information known as a metafile.

PHIGS supports the storage and manipulation of data in a centralized hierarchical data structure known as the Centralized Structure Store (CSS). The fundamental entity of data is a structure element; these entities are grouped into units called structures. Structures are organized as directed acyclic graphs called structure networks. The creation and manipulation of the data structure are independent of the display.

Graphical output on a workstation is produced by traversing a structure identified for display on the workstation and interpreting the structure elements. The workstation maps between four coordinate systems, namely:

- World Coordinates (WCs)—used to define a uniform coordinate system for all abstract workstations
- View Reference Coordinates (VRCs)—used to define a view
- Normalized Projection Coordinates (NPCs)—used to facilitate assemblies of different views
- Device Coordinates (DCs)—one coordinate system per workstation representing its display space

PHIGS supports two- and three-dimensional primitives, hidden line, hidden surface removal, and viewing transforms that support parallel and perspective transformations. PHIGS maps the graphical information that is input from a device as a result of operator actions into six classes of input, each represented by a data type referred to as a logical input value.

Appendix H
Ada Binding Products

VENDOR NAME	CDIF	GPSE	GPPF	GOSIP	GKS	IGES	IRDS	M_WIN	PHIGS	POSIX	SGML	SQL	TCP/IP	X_WIN	MOTIF	OPEN LOOK	X.25	X.400	X.500
Advanced Technology Center				P	A			P	A	P	P	P	P		A	P			
• AETECH									D	A									
• Aitech Systems, Ltd.										P				P					
• Alliant Computer Systems									A	A									
Alsys		A								P		P		D	A				
ASET								P	P	P					A	D			
• Compass												A							
• Concurrent Computer Corp.										A		D							
Convex Computer Corporation		A	A						P						P				
• Cray Research, Inc.										P		P	P	P					
• DDC International										A									
DDC-Inter, Inc.										P									
Digital Equipment Corporation	P	P	P	P	A/P	P	P	P	A/P	P	P	A/D	P		A/D				
• Encore		A	A							A		P	P		P	D		D	D
Hewlett-Packard					A				A	A/D		A		A/D	A/D				
• IBM										A		P		A					
• Informix												A							
Ingres Corporation												A							
Intermetrics												A							
Irvine Compiler					P				P					P					
MassTech		A	D																
• Meridian Software Systems, Inc.	A									A				A					
• MIPS Computwer Systems														D					
Objective Interface Systems, Inc.								D				D			A	D			
Oracle Corporation												A							
Quest Windows Corporation															D				
• Rational														A	P	A			
Rockwell International										D						P			
• Silicon Graphics									A					A/D					
• SKY Computers, Inc.										D				D					
SL Corporation		A	A		A				P	A		A	A		A	A	A	A	A
• STARS					A							A		A					
Software Technology, Inc.					A														
Sunrise Software International															A				
Systems Engineering Research Corp.															A				
• Tartan, Inc.										P					A	A			
TeleSoft					D				A	A		A	D						
U.S. Navy										P			D						
Verdix Corporation		P	P							P				A					
Visual Edge Software, Ltd.															D				
• York Software Engineering					A					A		A		D					

A = Binding is available in one or more configurations
D = Binding is being developed for one or more configurations
P = Binding is planned for one or more configurations

• = Information was collected by the U.S. Army Communications - Electronics Command (CECOM)

Appendix I
Lessons Learned

Historically, the Department of Defense (DOD) has failed to take advantage of past experiences in developing software systems. The process of reviewing past experiences and formulating new decisions based on these experiences should be ongoing. It must start from the beginning of the process and continue through development and into deployment. The software process also should be adjusted for overall system size, technical complexity, and development phase.

This appendix describes decisions and evaluations made about software system developments that led to lessons learned. Figure I-1 presents a matrix of the lessons learned by project in the following areas:

- Standards and policy
- Project management
- Development process
- Corporate knowledge and software development
- Training
- Resources and facilities
- Support environment and tools
- Reuse
- Project costs

In some cases, the information presented in the project descriptions came from internally developed evaluations; in others, the information is based on externally developed assessments. Note that, except for editorial changes, the descriptions are reproduced here as submitted by the originators.

As the examples show, different projects often experienced similar problems and applied similar remedies. Frequently, a series of non-software-related errors culminated in the production of the wrong set of software components. The lessons learned from these and other project experiences may help managers of new projects to avoid such problems.

Figure I-1. Lessons Learned Matrix (1)

Appendix Reference	Lesson	Standards/Policy	Project Management	Development Process	Corporate Knowledge/ Software Development	Training	Resources/ Facilities	Support Environment/ Tools	Reuse	Project Costs
I.1	**ADVANCED FIELD ARTILLERY TACTICAL DATA SYSTEM**									
1	Anticipate trouble with the Ada development tools/environment, no matter who is supplying them or when you get them.							X		
2	Budget for training.		X			X				X
3	Anticipate that original estimates for support hardware and facilities will have to be revised. Project experience resulted in a quadrupling of original estimates for support hardware and facilities.						X			X
4	Ensure that both the contractor and Government teams are knowledgeable about and understand the rationale for all software-related topics.		X	X	X					
5	Have the team develop a viable technical/management plan and adhere to it so that requirements and design can be implemented correctly.		X	X	X					
6	Report major problems up the line as encountered.		X	X	X					
7	Do not mistakenly blame software development for failure. Careful scrutiny of many projects frequently shows that things other than software development are responsible for failure.		X							

Figure I-1. Lessons Learned Matrix (2)

Appendix Reference	Lesson	Standards/Policy	Project Management	Development Process	Corporate Knowledge/ Software Development	Training	Resources/ Facilities	Support Environment/ Tools	Reuse	Project Costs
I.2	**AN/BSY-2**									
1	When externally supplied schedule constraints exist, the level of planning and execution analysis becomes much more critical.		X							
2	The greatest number of "lessons learned" is related to the contract requirements.		X	X	X					
3	Coordination frequently receives the least attention although it is one of the more important efforts.		X	X						
4	For large projects, it is mandatory that an adequately sized, qualified Technical Directive Authority Oversight Group be established and function for the duration of the project.		X	X	X					
5	Software development planning and monitoring must be done from the onset of Full-Scale Development and should take a phased approach (i.e., "build a little, test a little").	X	X	X						
6	Even the best made plans require changes during execution.		X	X						
7	For large efforts that are geographically disbursed, the goal should be to strive for commonality of development environment, tools, procedures, and product structure.		X	X	X		X	X		

Figure I-1. Lessons Learned Matrix (3)

Appendix Reference	Lesson	Standards/Policy	Project Management	Development Process	Corporate Knowledge/ Software Development	Training	Resources/ Facilities	Support Environment/ Tools	Reuse	Project Costs
I.3	**ADA LANGUAGE SYSTEM/NAVY**									
1	For DON Standard Embedded Computer Resources applications, top priority must be given to the real-time performance of the generated code.		X	X	X		X	X		
2	Even though actual software code production is only a relatively small portion of the total life cycle, it is critical to have a reasonable level of performance within the tool set.			X			X	X		
3	Each development effort should be managed under the assumption that there will be a formal production delivery to DON and a separate DON-controlled post-deployment phase.	X	X	X						
4	Requirements must be understood, and both formal and informal checks on the progress to meet these goals must be conducted throughout development.	X	X	X	X					
5	Because post-deployment support will be DON's responsibility, it is critical to build an adequate in-house team that is thoroughly familiar with the product before acceptance.	X	X	X	X	X				
6	The lack of full program funding commitment and support will have a negative impact on development plans. Be prepared to either alter the course and/or extend delivery schedules.		X	X						X
7	Producing a quality software-based product that meets its specified requirements is a difficult task.	X	X	X	X	X	X	X	X	X
8	No product is truly exercised and tested until it reaches the target user community.		X	X						

Figure I-1. Lessons Learned Matrix (4)

Appendix Reference	Lesson	Standards/Policy	Project Management	Development Process	Corporate Knowledge/ Software Development	Training	Resources/ Facilities	Support Environment/ Tools	Reuse	Project Costs
I.4	AVIONICS PROJECT									
1	Ensure that software production/cost modeling includes adequate time for the requirements/design phase before accepting externally generated completion dates.		X	X	X					
2	Be sure that requirements are fully defined and are traceable to test mechanisms.	X	X	X	X			X		
3	Do not plan to use equipment that is under development unless absolutely necessary. Apply a risk engineering approach to those items that must be used, place items on critical path, and monitor them closely.		X	X			X	X		
4	Always assume that everything could go wrong and perform full risk engineering and management.		X	X	X					X
5	Use a hands-on management approach from both the prime and Government perspectives, delineating clear lines of authority and responsibility for contractual requirements, especially for large projects.	X	X	X	X	X				
6	Specify in the contract requirements that capabilities must be established early, with adequate resources and authority. Closely monitor progress.	X	X	X	X	X	X	X		
7	Do not disregard the critical elements of the Military Standards (MIL-STDs) unless it is technically and managerially necessary to use alternative means. Develop a system-wide integration plan and follow it.	X	X	X	X					
8	Ensure that the schedule can accommodate slack and the possibility of DON independent test time for interim products and that the resources are available to support regression testing.		X	X	X		X			X

Figure I-1. Lessons Learned Matrix (5)

Appendix Reference	Lesson	Standards/Policy	Project Management	Development Process	Corporate Knowledge/ Software Development	Training	Resources/ Facilities	Support Environment/ Tools	Reuse	Project Costs
9	Do not disregard critical MIL-STD interim products in the contract requirements, and adequately plan for and execute the Government's role to ensure quality and delivery.	X	X	X				X		
10	Ensure that adequate development-support facilities exist by requiring these facilities to be contractually specified and monitored during the PRR process. Contingency plans should be available when and if problems develop.	X	X	X			X	X		
11	Do not let external schedule events drive the program. Develop input and output criteria for major milestones and adhere to them. It is very easy to build the wrong software.		X	X	X					
12	Where possible, use real production hardware and/or commercial prototypes to decrease the amount and scope of simulation.		X	X			X	X		
13	Ensure non-approval until requirements are met because when system requirements are not met and "as built" systems are approved, the contractor is no longer responsible for fixing the system.	X	X	X						
I.5	**PEO-SSAS, PMS-414, SEA LANCE**									
1	Use a consistent methodology throughout the program requirements, design, and coding phases to facilitate tracing requirements to the code.	X	X	X	X	X		X		
2	Use a common Program Design Language (PDL) across the project. On medium-to-large-scale systems, the PDL will contain a wide variety of differing coding techniques and code fragments.	X	X	X		X				
3	Include and enforce a requirement for a minimum ratio of 50/50 comments-to-code in the contract, software development plan, or coding guide.	X	X	X						

Figure I-1. Lessons Learned Matrix (6)

Appendix Reference	Lesson	Standards/Policy	Project Management	Development Process	Corporate Knowledge/ Software Development	Training	Resources/ Facilities	Support Environment/ Tools	Reuse	Project Costs
4	Use an automated format utility or equivalent software tool to ensure uniform code appearance. This can be applied by either quality assurance or configuration management.	X	X	X	X			X		
5	Develop a style guideline for the Ada code and PDL before doing any design work.	X	X	X						
6	Use software metrics from the beginning, and define basic terminology between Ada and the selected software development standard.	X	X	X						
7	Hammer out documentation requirements and licensing agreements between the Government and the contractors regarding the use of third-party software and the way it is to be tested and identified	X	X	X					X	X
8	Early in the development process, have the contractor provide a detailed list of tools that will be used in the development process and specify the format that will be used for transfer of source code, executable code, and software.	X	X	X			X	X		
L6	**NAVY WORLD WIDE MILITARY COMMAND AND CONTROL SYSTEM (WMCCS) SITE UNIQUE SOFTWARE (NWSUS) PROJECT**									
1	It is always safer to build and test incrementally.	X	X	X	X					
2	Planning for and designing in reuse yield long-term benefits.		X	X					X	
3	For large software undertakings, use of automated tools is mandatory.		X	X	X			X		

Figure I-1. Lessons Learned Matrix (7)

Appendix Reference	Lesson	Standards/Policy	Project Management	Development Process	Corporate Knowledge/ Software Development	Training	Resources/ Facilities	Support Environment/ Tools	Reuse	Project Costs
4	Until the design baseline has been approved and frozen, it is inadvisable to initiate with full-blown coding.	X	X	X	X					
5	If a risk engineering approach is taken to development (i.e., awareness, identification, technical/management of alternative solutions), then it is possible to undertake technologically challenging developments.		X	X	X					
I.7	**EVENT-DRIVEN LANGUAGE/ COBOL-TO-ADA CONVERSION PROGRAM**									
1	Training is essential for both technical and management personnel.		X	X		X				
2	Programmers require 4 to 9 months of training before they become proficient.		X	X		X				
3	Military transfers often result in a loss of investment in Ada training.		X			X				X
4	Systems originally written in languages that predate Ada that must be converted to Ada should be redesigned, not translated.		X	X	X					
5	Ada facilitates reuse.		X	X	X				X	
6	Ada lends itself to efficient code and high programmer productivity.		X	X						

Figure I-1. Lessons Learned Matrix (8)

Appendix Reference	Lesson	Standards/Policy	Project Management	Development Process	Corporate Knowledge/ Software Development	Training	Resources/ Facilities	Support Environment/ Tools	Reuse	Project Costs
7	Development tools are essential		X	X		X		X		
8	Development and maintenance time can be significantly reduced by applying software engineering principles and capitalizing on reuse.	X	X	X	X	X			X	
I.8	**SHIPBOARD GRIDLOCK SYSTEM WITH AUTO-CORRELATION**									
1	Before committing to large projects, the methods and tools to be used should be exercised. Quantitative evaluation of the expended resources should lead to better estimates for the work contemplated.	X	X	X			X	X		
2	The Ada code itself will have major architectural and design impact on a system; therefore, the two must be worked on simultaneously.		X	X						
3	A project should always try to build a little and test a little, building and testing the harder things first (e.g., system services and communications).	X	X	X						
4	A project should always attempt to involve the production hardware as early in the program as feasible.	X	X	X			X			
5	The team must be well trained in the use of supplied tools, and the tools must work as advertised.		X	X	X	X	X	X		
6	Adherence to good engineering practices is necessary when designing the system and its hardware and software.	X	X	X	X	X				

Figure I-1. Lessons Learned Matrix (9)

Appendix Reference	Lesson	Standards/Policy	Project Management	Development Process	Corporate Knowledge/ Software Development	Training	Resources/ Facilities	Support Environment/ Tools	Reuse	Project Costs
7	Until more technological progress is achieved, the potential for large-scale software component reuse is limited.		X	X					X	
I.9	**COMBAT CONTROL SYSTEM MK2**									
1	Ada experience and training are needed.		X	X		X				
2	Support software, practices, and products need constant attention.	X	X	X	X			X		
3	The need to interface with other language programs may place constraints on the type of Ada features that can be used.		X	X	X					
4	To ensure programming uniformity, a style guide should be developed and used across all developer teams.	X	X	X						
I.10	**P-3C UPDATE IV ADA DEVELOPMENT**									
1	Ada code requires more up-front time and effort, and the learning curve is slower.			X		X				
2	Increased facilities and memory are required to accommodate Ada code.		X	X			X	X		

Figure I-1. Lessons Learned Matrix (10)

Appendix Reference	Lesson	Standards/Policy	Project Management	Development Process	Corporate Knowledge/ Software Development	Training	Resources/ Facilities	Support Environment/ Tools	Reuse	Project Costs
3	Some software development tools are immature and have not been proven for many applications.				X		X	X		
4	Tailoring of DOD software development standards must be addressed to accommodate Ada-unique capabilities.	X	X	X						
5	Although the adoption of Ada was envisioned to enhance the software development process, use of Ada does not guarantee sound software engineering practice.		X	X	X	X				
6	Strict configuration management and control are required to enforce discipline to counter complexity-induced confusion.	X	X	X	X	X				
I.11	**STANDARD FINANCIAL SYSTEM REDESIGN**									
1	The available pool of developers skilled in Ada is limited.		X	X	X	X				
2	Few compilers and support tools are available to support information systems development in the IBM environment using Ada.		X					X		
3	Systems developed in Ada may be more maintainable than those written in COBOL.		X		X				X	
4	In principle, portability is ensured by developing code in Ada, but it is limited in practice.		X		X				X	

Figure I-1. Lessons Learned Matrix (11)

Appendix Reference	Lesson	Standards/Policy	Project Management	Development Process	Corporate Knowledge/ Software Development	Training	Resources/ Facilities	Support Environment/ Tools	Reuse	Project Costs
5	Ada has special advantages that make reuse more feasible and enables the benefits of reuse to be realized at several levels.		X	X	X				X	
6	Current documentation standards need to be reexamined.	X	X							
I.12	**RECONFIGURABLE MISSION COMPUTER PROJECT**									
1	For small technology demonstration projects, anticipate a lack of Ada compilers for small embedded computers that use advanced microprocessors.		X				X	X		
2	Plan on allocating a portion of the CPU utilization to the inefficiencies of using a modular design approach and design implementation in Ada.		X	X	X	X	X	X		
3	Plan on throwing away some or all of the first software designed.		X	X	X	X				
4	Dedicate an individual or group (depending on the size of the project) to the Ada hardware interface.		X	X	X		X			
I.13	**INTELLIGENT MISSILE PROJECT**									
1	System analysis must be conducted at the beginning to ensure adequate resources (compilers, hardware platform) will be available for meeting the system requirements.		X	X	X		X	X		
2	Even though Ada has many features, it does not have everything from every language; for example, Ada is restricted in the type of Artificial Intelligence systems for which it can be used.		X	X	X			X		

I.1 ADVANCED FIELD ARTILLERY TACTICAL DATA SYSTEM

The Advanced Field Artillery Tactical Data System (AFATDS) is a system of computers, printers, displays, and software that helps Army commanders plan, direct, and control artillery fire in combat situations. AFATDS was intended to replace the former Tactical Fire Direction (TACFIRE) system.

AFATDS was a concept evaluation effort that began in May 1984 with Magnavox Electronic Systems as the prime contractor. The paragraphs below summarize the lessons learned during this effort.

Lesson 1: *Anticipate trouble with the Ada development tools/environment, no matter who is supplying them or when you get them. Especially expect problems with the ability of the Ada Run-Time Executive to meet all of the project needs.* The Army had required Ada as the High Order Language (HOL). During the source-selection phase, only three validated compilers were available, none of which could down-line load to a target processor that met the AFATDS-derived requirements. The language, methodology, and tools were new; the approach was to be "software first."

Lesson 2: *Budget for training. Be prepared for and include additional funds for training over a long period of time. Note that for this training to be most effective, it must be accomplished just before or during the development effort.* Magnavox recognized that real-time expertise in Ada did not exist and immediately went to the Ada community to establish a comprehensive, long-term Ada and software engineering training program. Magnavox also proceeded to hire selected consultants and subcontractors to handle specialty items (e.g., database design).

Lesson 3: *Anticipate that original estimates for support hardware and facilities will have to be revised. In this project, original estimates quadrupled for support hardware and facilities.* Magnavox also purchased multiple mainframe and workstation computing systems; however, these resources proved insufficient but were relatively easy to upgrade.

Lesson 4: *To accomplish the project successfully, ensure that both the contractor and Government teams are knowledgeable about and understand the rationale for all software-related topics.* At that time, none of the DOD policy standards had been updated (this is still true today in many cases), and very few people on the Government side understood their ramifications. The Army had taken a sound, long-term view when it awarded this contract, but early into implementation, the pressure of outside scrutiny began to erode that resolve. Coupled with limited understanding of Ada and its software engineering ramifications, serious disconnects

began to develop between the contractor and the Army acquisition team (e.g., "where's the code" syndrome).

Lesson 5: *If the team develops a viable technical/management plan and remains steadfast, requirements and design can be implemented correctly. Although it will take longer to begin writing the actual code, it will be worth it because fewer design problems will be encountered during test and integration.* Some of the hardest work will be associated with trying to handle the external nay sayers.

Lesson 6: *Report major problems up the line as encountered.* Magnavox and the Army Program Office were never aggressive in promoting their initiatives. Had they been, many of the external groups might not have felt compelled to investigate, and more time would have been available to resolve the technical problems. Others can benefit from lessons learned only if people are informed about them. Such publicity could have helped the AFATDS project and provided insight to other projects that were beginning.

Lesson 7: *Do not mistakenly blame software development for failure. Careful scrutiny of many projects frequently shows that things other than software development are responsible for failure.* For AFATDS, three formal General Accounting Office (GAO) evaluations were performed and reported on during 1986-87: GAO/NSIAD-86-184FS, GAO/NSIAD-86-212FS and GAO/NSIAD-87-198BR. None of these reports identified Ada as a problem. Major impact items included the reduction in scope due to budget constraints, the changing of requirements to accommodate different equipment and software, and the Army's ability to manage this activity.

I.2 AN/BSY-2

The AN/BSY-2 Submarine Combat System (SCS) is the suite of hardware, software and equipment that will be used on the Department of the Navy's (DON's) next-generation attack-class submarine, the SSN-21. General Dynamics Electric Boat Division is building the first hull in this series, and the hull will be ready in 1994.

Lesson 1: *When externally supplied schedule constraints exist, the level of planning and execution analysis becomes much more critical.* This was especially true for BSY-2 due to the estimated volume of software and separately defined hull completion dates. The AN/BSY-2 software is being developed under DOD-STD-2167A in an effort that has combined aspects of the Concept Evaluation, Demonstration and Validation (D&V), and Full-Scale Development (FSD) phases of the life cycle. Commencing in 1985, a draft set of DON-generated SCS requirements was used for the System Design Definition (SDD) activity. Leading up to FSD and contract award, the two successful bidders, IBM and General Electric, worked with the Navy Team to solidify

requirements, develop design approaches, analyze ongoing prototyping efforts, identify critical items, fine tune the FSD Statement of Work (SOW), and generate three separate Source Lines of Code (SLOC) preliminary size estimates for the AN/BSY-2 System.

The other lessons learned on AN/BSY-2 fall into six distinct categories: contract, coordination, process, schedule, standards, and tools. Multiple lessons are presented for each of these areas. Note that the lessons do not apply exclusively to an Ada development and that they are presented randomly within each category (i.e., no attempt has been made to prioritize them).

Lesson 2: *The greatest number of "lessons learned" are related to the contract requirements.* The SOW should require regular reports on the status of all commercial products delivered as part of the system. This update should include information such as vendor, version number, performance statistics, licensing agreements, and plans for future modifications. In addition, when the same type of documentation is to be produced by multiple developers, implementation of a standardized style guide should be called out in the SOW. Furthermore, a provision should be included to allow deliverables to be transmitted in an electronic format. On systems that have classified information, installation and use of encrypted links between developer sites should be mandatory.

To ensure that requirements flow down adequately, the prime contractor should be required to provide copies and/or updates of all subcontract agreements to the acquisition agency.

To be fully effective, software quality assurance should be totally independent and organized to avoid a double chain of command (i.e., development program in the place of corporate quality assurance).

Identification, reporting, and close monitoring of available metrics should begin early in development. The level of detail should increase in tandem with advanced development. Metrics should be analyzed thoroughly, and results should be incorporated into quarterly program assessments. Progress or regression relative to the program plan baseline should be a key element in this assessment. Separate analyses conducted by DON for comparison purposes produced additional benefits for AN/BSY-2 when results of these analyses were shared with the developer.

To ensure that the metrics data received are comparable across all development teams, a uniform SLOC counting methodology must be defined and followed.

Lesson 3: *Coordination frequently receives the least attention although it is one of the more important efforts.* Early in the contract, direct lines of communication should be established among key participants: acquisition agency, developer, technical agency, Independent Verification and Validation (IV&V) agency, quality personnel, and Commercial-Off-The-Shelf (COTS) software vendors. Such "shortcut" communiques result in more efficient problem identification and resolution, which have an overall positive effect on cost and schedule.

Informal networking among groups of like interest will increase the effectiveness of each group. Regularly scheduled communication tends to short-circuit problems while providing a broader perspective to participants. For example, AN/BSY-2 holds a monthly user group meeting to discuss problems, workarounds, and successes with the operating system. The vendor's active participation at these meetings has increased responsiveness to and visibility of AN/BSY-2 needs.

The prime contractor should maintain tight control of subcontractor efforts through weekly monitoring and quarterly audits. Furthermore, attendance at technical and working group meetings should be mandatory for all team members.

Lesson 4: *For large projects, it is mandatory that an adequately sized, qualified Technical Directive Authority (TDA) Oversight Group be established and function for the duration of the project.* Very early in development, the contractor should detail each process proposed for use in the program. These processes should be defined in approved, baselined documentation. (Data Item Descriptions [DIDs] need to include more stringent, detailed guidelines.) Multidisciplinary contract agency representatives should then closely review each process in software development and in related areas (configuration management, quality assurance, test) for adequacy, consistency, and completeness. Contractor modifications to these processes should be presented during formal reviews and entered into the baseline document only upon approval.

A streamlined waiver request process should be established for reporting proposed contract deviations to language and/or contract requirements. Waiver packages should be initiated every 6 months, depending on program size and life span.

A comprehensive Ada training program should be developed to address application-specific requirements. This program should be capable of transitioning seasoned engineers yet flexible enough to instruct entry-level programmers.

Ada methodologies (e.g., exception handling) should be defined early in development. Partial tasking should be considered as an alternative for reducing

rendezvous time. Establishing global error models well in advance of detailed design will result in a more robust system.

Ada guidelines and procedures should be established primarily by the program's resident Ada experts. These lessons learned should be provided in an Ada style guide as an appendix to the software Standards and Procedures Manual. For example, compilation dependencies can be reduced and debugging smoothed by avoiding subprogram nesting. This think tank of Ada experts should also be convened to resolve complex, persistent, Ada design problems. For example, enhancement of time-critical processes can be effected through expert application of Rate Monotonic Scheduling techniques.

Lesson 5: *Software development planning and monitoring must be done from the onset of FSD and should take a phased approach (i.e., "build a little, test a little").* Ada software development schedules should allow for longer requirements and design phases and shorter test and integration phases. The schedule should contain Critical Design Reviews (CDRs) to correspond to the incrementally developed software. In addition, testing should occur using manageably sized units at phased steps with explicit success criteria.

The delivery schedule for software plans, standards, and procedures should show compressed early deliveries. Multiple early drops should accelerate establishment of a baseline. These planning documents should be baselined and under formal configuration control no later than close of the preliminary design phase. Conversely, software requirements specifications should have fewer drops, a longer document review cycle, and a baseline before preliminary design.

Product Readiness Reviews (PRRs) should be held early in development. These reviews have a positive, cohesive effect and provide a close, systemwide look at processes, products, personnel, and facilities. Implementation of an action item system is key to PRR effectiveness.

The developer should identify critical-path software items (e.g., shared system services). Close management of this process should ensure early delivery and test of these functions.

Lesson 6: *Even the best made plans require changes during execution.* AN/BSY-2 used DOD-STD-2167A for software development guidance. The intent of this standard, however, is to provide a software development superset from which extraneous requirements can be eliminated. AN/BSY-2 staff carefully tailored this standard, mindful that it is easier to provide relief from requirements than to "buy" them in later. The contracting office should remain open to negotiations on tailoring

DOD standards as phases unfold, technology advances, and/or lessons are learned. As an example, support software documentation has been reduced from the full suite to design notebooks and operator/maintenance manuals.

As part of tailoring the standards, a cross-check should be performed against the SOW. Checking requirements in the SOW for potential ambiguities or even conflicts within the military standards may avoid costly rework during later phases.

Lesson 7: *For large efforts that are geographically disbursed, the goal should be to strive for commonality of development environment, tools, procedures, and product structure.* The contracting agency should require standardization of support tools across the program. Although the up-front cost is greater, long-term benefits gained from such commonality make it a worthwhile investment. Use of common tools allows problems to be identified and workarounds made only once and results entered into a shared electronic reporting system. In addition, data exchanges among development teams are less time-consuming and more efficient, thus reducing the risk of error.

For large projects, it is imperative that the configuration management system be capable of supporting rapid turnaround during the integration and test phase. The system should provide configuration management of all software support tools as well as the development code. In addition, a version control process must be established and enforced by the prime contractor for these tools.

A common database should be established to electronically track requirements down through software requirements specifications and hardware unit specifications and, later, into test. Use of this method will enhance traceability and ensure flowdown of requirements. A common database also should be created to track connectivity of software interfaces. Consistency checks should be run for early detection of misaligned interfaces.

Commercial support tools may require modifications to handle large Ada developments, and non-Ada commercial code slated for incorporation into the product may create interface and performance problems. Additional time and resources should be factored into development plans to allow for these potential stumbling blocks. (Computer resources also should be supplemented to account for the increase in demand that traditionally occurs during Ada developments.)

Compiler benchmarks should be evaluated before compiler selection is finished. (Compilation time should be factored in as an additional consideration.) The developer should know the weaknesses as well as the strengths of the Ada constructs (link library sizes, nesting of generics, etc.) as used in the compiler and/or Ada

Programming Support Environment (Ada PSE). Binding approaches should be established and benchmarked early in the development.

Use of an Ada standards checking tool is highly recommended. Using a standards checker not only encourages production of high-quality code but also reduces manpower efforts and enhances maintainability.

I.3 ADA LANGUAGE SYSTEM/NAVY

The Ada Language System/Navy (ALS/N) FSD program implements Ada for use with DON's standard embedded computers: AN/UYK-43, AN/UYK-44 and the P3I AN/AYK-14. Since January 1989, DON has mandated the use of ALS/N as the first-line support software consideration for the DON standard processors. Although ALS/N is a support software effort, it also is a large software-based systems development effort. The ALS/N development project has produced more than 1 million lines of Ada code that also support DOD-STD-2167A documentation.

The DON Ada Standard Embedded Computer Resources (SECR) effort began in the early 1980s and closely monitored the other Service efforts, such as the Army Ada Language System (ALS) effort and the Air Force Ada Integrated Environment (AIE) effort. The DON goals were to avoid reinventing the wheel and to maximize the benefits of the Ada reuse/portability concepts for developing support software. In 1984, DON opted to establish the baseline with the Army ALS and proceeded to develop specific SECR-retargeted compilers and tools.

Lesson 1: ***For DON SECR applications, top priority must be given to the real-time performance of the generated code. Performance requirements must be formally specified, and performance capabilities must be tested before product acceptance and deployment.*** Due to the number and severity of the problems encountered, the Army paid little attention to performance issues for the support environment and the targeted real-time environment.

Lesson 2: ***Although actual software code production is only a relatively small portion of the total life cycle, it is critical to have a reasonable level of performance within the tool set. At a minimum, the tool set must meet both programmer functional and CM needs.*** The Army ALS tool set had been implemented in Ada but operated on the VAX/VMS host environment through an additional layer called the Kernel Ada Programming Support Environment (KAPSE). This arrangement made tool performance unacceptably slow. The Navy, therefore, redirected the contractor to eliminate the KAPSE requirement.

Lesson 3: ***Each development effort should be managed under the assumption that there will be a formal production delivery to DON and a separate DON-controlled***

post-deployment phase. To ensure continuous development oversight, DON laboratory personnel were provided to facilitate the transition to life-cycle support.

Lesson 4: *Requirements must be understood, and both formal and informal checks on the progress to meet these goals must be conducted throughout development.* More than 15% of the budget was expended on an independent test team, an approach used by the Air Force for its effort. This team performed TDA-type testing that included full knowledge and understanding of the product internals. Concurrently, a separate IV&V agent performed "black box" testing to evaluate formally the specified requirements. Expenditures for this support were approximately 5% of the total budget.

Lesson 5: *Because post-deployment support will be DON's responsibility, it is critical to build an adequate in-house team that is thoroughly familiar with the product before acceptance.* The ALS/N development has actively funded various Navy laboratories (e.g., Naval Surface Weapons Center [NSWC], Naval Avionics Center [NAC], Naval Undersea Command [NUSC], Naval Air Development Center [NADC], and Naval Ocean Systems Center [NOSC]) to participate in the program and also involved the Navy's life-cycle agent (i.e., Fleet Combat Direction System Support Activity [FCDSSA], San Diego).

Lesson 6: *Lack of full program funding commitment and support will have a negative impact on development plans. Be prepared to either alter the course of and/or extend delivery schedules. Always try to maintain the best possible product quality and maximize life-cycle supportability within the program constraints.* The vagaries of year-to-year funding support tend to disrupt large undertakings that involve many elements such as laboratories, prime contractors, subcontractors, IV&V, and independent test organizations. All parties have to be motivated, good informal communication mechanisms must be in place, and all development efforts must be carried out according to an agreed-to plan that can accommodate a certain degree of flexibility.

Lesson 7: *Producing a quality software-based product that meets its specified requirements is a difficult task.* ALS/N provides a software means to upgrade deployed SECR processor-based systems indefinitely. ALS/N also can be considered as the front-line consideration for new systems developments because DON has 100% ownership or change control rights. Many U.S. commercial companies provide Ada compiler technology. Investment costs for those technologies that have been commercially successful are consistent with DON expenditures for ALS/N. However, few of these commercial Ada technologies specifically addressed real-time performance to the degree of ALS/N capabilities, which is required for Mission-Critical Computer Resources (MCCR) applications. In fact, two out of every

three DON dollars have been spent on DON standard Run-Time Environment needs. The ALS/N FSD program has produced compilers and run-time operating systems that will meet many of the performance requirements as specified.

Lesson 8: *No product is truly exercised and tested until it reaches the target user community. It is best to phase systems into deployment via beta testing and friendly users before public release.* Currently, four DON Research and Development (R&D) centers use ALS/N in a test and evaluation mode. The DON MCCR waiver process now includes ALS/N consideration as part of the standard acquisition formula for both new starts and upgrades.

I.4 AVIONICS PROJECT

The avionics project is a major system upgrade for an airborne Command, Control, and Intelligence (CC&I) application that targets existing platform and potential forward fit into next-generation aircraft. The upgrade is to improve acoustic and nonacoustic processing capabilities as well as signal processing, detection and classification, multistation integrated systems, data buses, and communications.

Lesson 1: *Ensure that software production/cost modeling includes adequate time for the requirements/design phase before accepting externally generated completion dates.* The contract was awarded in July 1987 with a prototype scheduled for delivery in July 1990. An optimistic schedule of 1.2 million SLOC is projected.

Lesson 2: *Be sure that requirements are fully defined and are traceable to test mechanisms. Include necessary Government visibility into the process. Beware of shortcuts and bad engineering practices, especially when there is a prime contractor-subcontractor team organization.* The Firm Fixed-Price (FFP) contract included production options. The contract options were tied to calendar exercise dates, without a requirement to demonstrate performance capabilities.

Lesson 3: *Do not plan to use equipment that is under development unless absolutely necessary. Apply a risk engineering approach to those items that must be used, place items on critical path, and monitor them closely.* The contract included the planned use of "in-development" Government-Furnished Equipment (GFE) and Contractor-Furnished Equipment (CFE).

Lesson 4: *Always assume that everything could go wrong and perform full risk engineering and management.*

Lesson 5: *Use a hands-on management approach from both the prime and Government perspectives and delineate clear lines of authority and responsibility for*

contractual requirements, especially for large projects. In addition, do not take a hands-off approach to subcontractor management.

Lesson 6: *Specify in the contract requirements that capabilities must be established early, with adequate resources and authority. Closely monitor progress.* A plan must be developed for handling distributed development environments/deliverables exchanges. Such planning must have been contractually required and completed, and it must receive some degree of Government approval/monitoring before the program is executed. A "sell off" from a subcontractor to the prime contractor must address all contingencies when the prime contractor-to-DON delivery requires changes, retesting or documentation, and the like. Configuration management and quality assurance should be standardized and coordinated across the whole effort. Formal, standardized software development procedures should be specified in the contract and approved before being implemented. Lack of such formal, standardized procedures cannot be condoned, especially across larger projects. The procedures should be monitored to ensure that the documented process is being implemented.

Lesson 7: *Do not disregard the critical elements of the Military Standards (MIL-STDs) unless it is technically and managerially necessary to use alternative means. Develop a systemwide integration plan and follow it.* During the development of the avionics project plan, a systemwide integration plan was not developed.

Lesson 8: *Ensure that the schedule can accommodate slack and the possibility of DON independent test time for interim products. Also ensure that the resources are available to support regression testing.* The avionics schedule contains no plan for slack or for resources to support regression testing.

Lesson 9: *Do not disregard critical MIL-STD interim products in the contract requirements, and adequately plan for and execute the Government's role to ensure quality and delivery.* Mutually agreed-to criteria for major milestones must be met, or action item work plans must be created for unmet criteria.

Lesson 10: *Ensure that adequate development support facilities exist. Existence of these facilities should be contractually specified and monitored during the PRR process. Contingency plans should be available when and if problems develop.* Inadequate facility estimates, combined with no forward-looking projection analysis and unavailability of contingency plans, resulted in severe problems as the interim product grew in size.

Lesson 11: *Do not let external schedule events drive the program. Develop input and output criteria for major milestones and adhere to them. It is very easy to build the*

wrong software. During the avionics project, time spent in the requirements/design phase was insufficient to mature the software baseline.

Lesson 12: *Where possible, use real production hardware and/or commercial prototypes to decrease the amount and scope of simulation.* The simulator software must be treated as critical-path material if it is to be used during development. Simulator software also should be documented as operational software because it will be critical when mission requirements are being tested. (For example, the system may function in a simulator environment but fail in the real world.)

Lesson 13: *When system requirements are not met and "as built" systems are approved, the contractor is no longer responsible for fixing the system.* The system should not be approved until requirements are met. Design information should not be placed in Software Requirements Specifications (SRSs) and Interface Requirements Specifications (IRSs).

I.5 PEO-SSAS, PMS-414, SEA LANCE

The SEA LANCE Anti-Submarine Warfare Standoff Weapon (ASWSOW) was being developed to provide Vertical Launching System surface combatants and nuclear power attack submarines with a standoff-range missile for use against hostile submarines. Before partial program termination in December 1989, the program was in FSD.

The SEA LANCE is a long-range Anti-Submarine Warfare (ASW) missile system developed to complement ship-launched torpedoes and helicopter-borne weapons by providing a quick-kill opportunity at long ranges. The SEA LANCE also can be launched in a buoyant protective capsule that floats to the surface from a submarine torpedo tube. The tactical missile employs seven embedded processors for providing guidance, navigation, and flight control functions. These tactical processors are the Guidance Electronics Unit (GEU), which uses a Motorola 68020/68881 processor; the Inertial Measurement Unit (IMU), which uses a Zilog Z8002 processor; the Pulse Driver Unit (PDU), which uses an INTEL 8797 processor; and four Fin Actuator Units (FAUs) each of which uses an INTEL 8797 processor. Software has been developed under the guidelines of DOD-STD-1679 for each of these subsystems, the most extensive development effort being for the Guidance, Navigation, and Control Program (GNCP) in the GEU.

SEA LANCE system software consists of the embedded GNCP; three embedded small systems software programs (IMU, PDU, FAU); two embedded instrumentation/flight termination system programs; and missile test set, support, simulation, and adaptor/interface electronics software. Ada was used as the Program Design Language (PDL) and the high-order implementation language only for the

development of the GNCP. The following languages were used in all of the other SEA LANCE software development efforts: IMU—Z8000 Assembly; PDU—PL/M 96; FAU—PL/M 96; Arm and Control Unit—PL/M 96; Instrumentation Data Unit—68020 Assembly; missile test set software—Pascal; support software—Pascal, FORTRAN, and Assembly; simulation software—FORTRAN and specialized languages. All discussion and lessons learned are concerned only with the GNCP.

The GNCP is a digital computer program totally contained in nonvolatile memory, which resides in the missile's GEU. It consists of approximately 20,000 SLOC (100,000 Physical SLOC). The GNCP was being developed in accordance with the guidelines of DOD-STD-1679 using the VERDIX Ada Development System (VADS). Before program termination, the GNCP had successfully passed through program milestones such as Preliminary Design Review (PDR) in August 1984, a Delta-PDR in February 1988, an In-Process Review (IPR) in March 1989, and numerous Technical Interchanges between 1983 and 1989. Draft versions of a test specification, test plan, and test procedures were developed in parallel to the design. The GNCP was developed, tested, and integrated at the module and system level in the contractor's Computer Program Development Laboratory (CPDL), Operational Mock-Up (OMU) Laboratory, and System Integration Laboratory (SIL). Performance and most preflight testing of the GNCP was done in the SIL to fully exercise each function specified by the performance specification. The GNCP guided the test missiles along two near-perfect trajectories in the only two SEA LANCE Contractor Test and Evaluation flight tests in February 1990.

Because the GNCP had not yet reached CDR at the time of program termination, DON never approved or accepted it. As part of the partial termination efforts, the GNCP design of record was documented in accordance with DON direction and archived.

As part of the partial termination efforts, a DON/Boeing study is in process. This study shows the impact of switching to the newer defense software development standards (DOD-STD-2167A and DOD-STD-2168). The study is being conducted in accordance with the guidelines of Military Handbook (MIL-HDBK)-287.

Lesson 1: ***Use a consistent methodology throughout the program requirements, design, and coding phases to facilitate tracing requirements to the code.*** SEA LANCE used a functional decomposition method in developing the requirement specifications, then used an Object-Oriented Design (OOD) methodology when developing the design specification and the code. The two methods had to be combined. Because SEA

LANCE was a fire-and-forget weapon, the traceability of every performance requirement was considered extremely important. Use of two design methods made it difficult to trace the requirements from the Performance Specification into the Design Specification and then into the code itself.

Lesson 2: *Use a common PDL across the project. On medium to large-scale systems, the PDL will contain a wide variety of differing coding techniques and code fragments.* SEA LANCE used Ada as its PDL. It was learned that when using Ada as a PDL, the software development and uniform coding standards should be enforced on the PDL as well as the actual Ada code.

Lesson 3: *Include and enforce a requirement for a minimum ratio of 50/50 comments-to-code in the contract, software development plan, or coding guide.* Although Ada is more readable than many other languages, it still requires a liberal use of comments to describe what is going on and why. Generally, Government code reviewers needed more review time because of the lack of comments.

Lesson 4: *Use an automated format utility or equivalent software tool to ensure uniform code appearance. This can be imposed through either quality assurance or configuration management.* The SEA LANCE contractor did not always use a printer format utility or other automated tools to ensure uniform appearance of the code. As a result, many Ada specifications and bodies had a unique appearance, depending upon the individual coder.

Lesson 5: *Develop a style guideline for the Ada code and PDL before doing any design work.* The SEA LANCE contractor developed most of the PDL without a formalized Ada coding guideline. The result was a PDL that sometimes differed from module to module in appearance, style, and coding format.

Lesson 6: *Use software metrics from the beginning and define basic terminology between Ada and the selected software development standard.* The minimal use of software metric tools and the defining of basic terms in the early development process gave rise to conflicts between the contractor and the Government as to what constituted a module, a line of code, or the difference between a PDL line of code and an operational line of code.

Lesson 7: *Hammer out documentation requirements and licensing agreements between the Government and the contractors regarding the use of third-party software and the way it is to be tested and identified.* The SEA LANCE contractor employed a proprietary third-party Ada Run Time Executive, and the Government had trouble obtaining documentation on the inner workings and testing of the Run Time Executive software.

Lesson 8: *Early in the development process, have the contractor provide a detailed list of tools that will be used in the development process for the PDL/code and specify the format that will be used for transfer of source code, executable code, and software documentation to the Government. (Note that DOD-STD-1679 did not require a Computer Resource Integrated Software Document [CRISD].)* The Government had some difficulty finding compatible computers to load in contractor-transferred software listings. It also proved difficult to identify the exact format of software deliverables and the exact configurations of the contractor-used development tools.

I.6 NAVY WORLD WIDE MILITARY COMMAND AND CONTROL SYSTEM (WWMCCS) SITE-UNIQUE SOFTWARE (NWSUS) PROJECT MISSION

Lesson 1: *It is always safer to build and test incrementally.* Space and Naval Warfare Systems Command (SPAWAR) PMW 161-5 is responsible for modernizing eight existing site-unique COBOL 1968 applications with approximately 300K of Ada source code on the NWSUS project. These applications are operational on the WWMCCS Honeywell DPS8 mainframe and are being reengineered using Ada OOD with DOD-STD-2167A because Honeywell is phasing out maintenance of COBOL 1968. This is within the WWMCCS Automatic Data Processing (ADP) Modernization (WAM) Program. The NWSUS project, which is divided into three increments, is in the 3d year of a 5-year effort. The first increment consists of six smaller applications with the larger applications in the later increments.

Lesson 2: *Planning for and designing in reuse yield long-term benefits.* The project is in accordance with DOD-STD-2167A/2168 tailored for OOD. The existing COBOL applications are used to capture requirements. Development is performed on a Rational R-1000 model 40 with Honeywell DPS8 and IBM PC/XT clones as targets. With one exception, the applications are management information systems (MISs), and the development makes extensive use of a common set of reuse components.

Lesson 3: *For large software undertakings, use of automated tools is mandatory.* The 2167A documentation is being developed on the Rational, and a Computer-Aided Software Engineering (CASE) tool has been developed to validate the completeness and consistency of the requirements, design, object/class specifications, and Ada specifications. Two "4GL-like" productivity tools, used in conjunction with the reuse components to create application screens and reports, are used for rapid prototyping and to support the generation and standardization of the user interface.

Lesson 4: *Until the design baseline has been approved and frozen, it is inadvisable to initiate full-blown coding.* An initial CDR was completed for Increment 1 in April 1990, and a second CDR to review redesign caused by a change of target is being conducted this spring. Development of many of the reuse components has been

completed, and an operational prototype of one of the applications will be completed by CDR. Full development of the Increment 1 configuration items has begun and is scheduled for completion in FY 1992.

A full Object-Oriented Requirements Analysis (OORA) and specification for the Increment 2 configuration items were completed at the System Design Review (SDR), which was very successful. Both the site customer and SPAWAR commented on the effectiveness of OORA. The CDR occurred in October 1991.

Lesson 5: *If a risk engineering approach (i.e., awareness, identification, technical management of alternative solutions) is taken to development, then it is possible to undertake technologically challenging developments.* Conventional wisdom has it that a project with a new application area, a new programming language, or new personnel will have trouble. NWSUS had all three; consequently, the project has had its share of problems. The problems spanned development methodology and standards, target development environment (both Ada compiler problems and problems with the compiler/operating system bindings), Ada training and startup, software reuse, contract structure, and management. However, NWSUS has managed to survive these problems and is currently in a productive mode.

The following lists some of the problems encountered and their solutions or workarounds.

Problem	Resolution
Ada compiler was unavailable for Honeywell DPS8, and WWMCCS Information System (WIS) Workstation target was unavailable at contract start.	The Rational was selected as the host development environment for all applications. Testing is first done on the Rational and then on the target.
Functional analysis was required for the first increment.	The functional analysis approach did not work out well. Full object-oriented analysis was used for the second increment, and that approach has been very beneficial.

Contract assumed that all configuration items were the same, and a hard split between design and code hindered Ada OOD.	The contract structure was modified to reflect the diversity of the configuration items and the R&D nature of the project and to allow an efficient mechanism for reuse components and prototyping.
Contract and management of reuse between applications initially was weak and/or missing.	An internal approach was used to support reuse on a level-of-effort Work Breakdown Structure (WBS). DON recognized the need in the contract update.
DPS8 Ada compiler was not mature and late; the WIS Workstation was canceled.	The workstation target was changed to a personal computer (PC). Redesign is under way for the new target and the problems encountered with the DPS8 Ada compiler.
Initial training was affected by the "3-week syndrome."	Initial training was too compressed and not project specific. NWSUS now uses a part-time, 2-month, in-house training seminar with a "lab session" that uses project deliverables.
OOD proved to be labor intensive during the first increment.	Ada OOD proved to be a very effective development approach because it gives much more visibility and control of the analysis and design. The downside is that this requires much more effort. We found no available CASE tools that supported it, and too much had to be done manually. The validation process was automated for the second increment.

I.7 EVENT-DRIVEN LANGUAGE/COBOL-TO-Ada CONVERSION PROGRAM

From 1987 to 1989, the Marine Corps replaced its aging inventory of ruggedized IBM Series-1 minicomputers with hardened IBM-compatible microcomputers. The transition required that all of the systems originally programmed for execution on the Series-1 be ported to the microcomputer. Approximately 25 systems were written in Event-Driven Language (EDL) or COBOL. At about the same time, Ada was introduced as the standard programming language for DOD. The close proximity of the two events provided the Marine Corps with an opportunity to gain valuable expertise in the new DOD standard programming language through reverse engineering of well-known systems. At the time, the Marine Corps had no organic Ada programmers and no expertise in its associated design methodologies.

The reprogramming effort was divided among three Marine Corps Central Design Programming Activities (CDPAs) along functional boundaries. In the process of the reprogramming effort, the Marine Corps learned several lessons.

Lesson 1: *Training is essential for both technical and management personnel.* To take full advantage of Ada, designer/analysts must be familiar with the principles of software engineering and the way Ada supports those principles. Because few Marines had knowledge of Ada design methodologies at the outset, the tendency was to recode the original system designs in Ada. The original system designs were often derived directly from the existing EDL/COBOL code. Because neither of those languages contains all of the Ada constructs, the advantages of Ada did not always materialize.

Lesson 2: *Programmers require 4 to 9 months of training before they become proficient.* It takes 4 to 9 months of formal and on-the-job training before a programmer becomes proficient in Ada. However, after that initial training period, the programmer should be capable of producing code very rapidly when given a good design and programming library.

Lesson 3: *Military transfers often result in a loss of investment in Ada training.* Because proficiency in Ada can take as much as 9 months to attain, a newly trained programmer is productive only for a portion of his or her tour. Unless steps are taken to ensure reassignment to another Ada shop, the training investment is likely to be lost.

Lesson 4: *Systems originally written in languages that predate Ada that must be converted to Ada should be redesigned, not translated.* After the first few projects, it was evident that inefficiencies in the original designs were being duplicated in the Ada translations.

Lesson 5: *Ada facilitates reuse.* During the conversion effort and on subsequent projects, the Marine Corps found that on an average project, only 45% of the code had to be written from scratch; the other 55% came from reuse. Reusable code generally came from previous projects and development tools (e.g., AdaSAGE). In recent projects, the Marine Corps has consulted Ada software repositories for reusable code in an effort to reduce development time and effort wherever possible.

Lesson 6: *Ada lends itself to efficient code and high programmer productivity.* The syntactical structure of Ada helped the Marine Corps implement many of the software engineering principles. Modularity, information hiding, localization, and abstraction were easily implemented.

Lesson 7: *Development tools are essential.* Initially, lack of a good tool kit hindered the conversion effort. In-house tools were built to overcome Ada file limitations and to enhance screen management. Shortly thereafter, the Marine Corps funded the development of AdaSAGE, which reduced development time by as much as 50%.

Lesson 8: *Development and maintenance time can be significantly reduced by applying software engineering principles and capitalizing on reuse.* The Marine Corps estimates that from 15% to 60% reduction in development and maintenance time are being achieved when software engineering principles and reuse are applied.

I.8 SHIPBOARD GRIDLOCK SYSTEM WITH AUTO-CORRELATION

The Shipboard Gridlock System with Auto-Correlation (SGS/AC) application plays a fundamental role in the coordination of multiplatform shipboard systems by processing own-ship and remote track data within a common positional frame of reference. This application performs gridlock processing to correct for sensor and navigational errors while correlating the identified tracks from remote systems. This software-based application is characterized by hard deadlines; multiple external interfaces; and time-critical, computationally intensive processing. The SGS/AC is deployed on the Aegis cruiser/destroyer class of surface ships.

Lesson 1: *"Fly before you buy." Before committing to large projects, the methods and tools to be used should be exercised. Quantitative evaluation of the expended resources should lead to better estimates for the work contemplated.* This project is being performed by the Naval Surface Warfare Center (NAVSWC). It can be characterized as a D&V development effort that parallels the SGS/AC program implemented in Compiler Monitor System-2 (CMS-2) for either the AN/UYK-20 or the AN/UYK-44 target processors. This parallel effort uses ALS/N as the host development tool set and targets an AN/UYK-44 processor configuration. An additional objective of the effort is to generate a comprehensive comparative analysis

of the CMS-2 and Ada developments that includes quantitative data and information pertinent to future Aegis-class combat direction system upgrades.

Lesson 2: *The Ada code itself will have major architectural and design impact on a system; therefore, the two must be worked on simultaneously.* From the outset, it was recognized that to simply translate CMS-2 code to Ada would be technically feasible but would not produce any long-term benefit.

Lesson 3: *A project should always try to build a little and test a little, building and testing the harder things first (e.g., system services and communications).* The new design effort attempted to minimize the run-time overhead, include portability in the design, manage interfaces to get best-case response under worst-case loads, and maximize robustness and predictability. A multiphased build plan was initiated.

Lesson 4: *A project should always attempt to involve the production hardware as early in the program as feasible. Successful simulator and emulator runs mean nothing when the delivered code does not work on the real hardware. Acceptance requirements must be set correctly, or development schedule reserve must be allocated to absorb such difficulty. Things will go wrong, and this should be anticipated.* All development is being carried out on VAXs, with DEC Ada being used during the early code and test phases. The target AN/UYK-44 processor requires special cards to run the Ada code. The particular configuration was unavailable until well into the project.

Lesson 5: *The team must be well trained in the use of the supplied tools, and the tools must work as advertised.* The ability to fully define a working set of integrated tools early in development and to acquire them as they are needed is critical. For example, a symbolic debugger is an absolute necessity.

Lesson 6: *Adherence to good engineering practices is necessary when designing the system and its hardware and software.* Although this project is a relatively small software undertaking, establishing and enforcing sound software design methodology and development processes, such as coding standards, documentation production, and code reviews, help overcome lapses in memory, personnel turnover, lack of focus, and lack of requirements to trace verified design/code.

Lesson 7: *Until more technological progress is achieved, the potential for large-scale software component reuse is limited.* This project has shown that achieving real-time developments requires meeting hard deadlines and getting close to the target machine, which often conflicts with the concept of code component reuse.

I.9 COMBAT CONTROL SYSTEM MK2

The Submarine Combat Control System (SCCS) Program focuses on consolidating the various Combat Control and Defensive Weapon Systems (DWSs) software configurations that are in use on deployed SSN-688 and SSN-726 class submarines. These vessels constitute both the defensive (attack) and strategic platforms for the DON submarine force. The SCCS upgrade will either upgrade or replace obsolete general-purpose computers, peripherals, display consoles, and weapons simulators. This software upgrade provides a common software package for both classes of submarine and incorporates operational and maintenance-related enhancements. The SCCS Program also includes the development of systems to support crew training and land-based testing.

The software for the SCCS consists of new development software and firmware, modified Government-Furnished Software (GFS) and firmware, and unmodified commercial software and firmware.

Most of the modified GFS software has been written in either DON standard CMS-2 HOL or the ULTRA-32 Assembler. The project mission is to develop a maintenance capability that improves the chances for coordinating evolutionary change in these shipboard systems.

The new portion of the CCS MK2 program involves integrating a replacement man-machine interface display console and associated Ada application software into the existing deployed systems. The approximate language mix is as follows:

Language	SLOC
CMS-2 & ULTRA-32	2M (GFS/Modified)
Ada	581K (new)
C	279K (commercial)
FORTRAN	149K (retained)

The Ada SLOC are being developed under DOD-STD-2167A requirements. The CMS-2, FORTRAN, and ULTRA-32 software were all developed under DOD-STD-1679A.

The paragraphs below summarize the lessons learned about Ada on this project.

Lesson 1: ***Ada experience and training are needed.*** The majority of experienced personnel in this defense area had little or no experience with Ada and modern software engineering practices. It was necessary to evaluate bidders on their in-place Ada expertise and on their ability and/or plans to acquire or build on that base. To properly monitor or manage the development, in-house capabilities had to be built

up in these areas. It is especially important to use hands-on training as close to development as possible or during development.

The relative immaturity of candidate Ada products, coupled with the specific need to handle many foreign language interfacing requirements, meant that the developer team needed a very close relationship with their candidate Ada development tool suppliers.

Lesson 2: *Support software, practices, and products need constant attention.* This undertaking required that the chosen contractor be capable of using automated tools to manage and technically execute this large programming development. To that end, source selection criteria were established and used during the source selection process.

Each project has to generate its own Computer Resources Life-Cycle Management Plan (CRLCMP) and Integrated Logistic Support Plan (ILSP) before the Defense Acquisition Board (DAB) Milestone I. However, unless the Government defines the total development environment fully and requires its use as part of the proposal, difficulty will ensue as differences develop between the methodology, tools, and equipment used by the developer and those specified by the Program Office. Typically, the parties involved will have opposing agendas. Coupled with the inability of many tools to scale up to programming in the large or even to exchange data structures efficiently, this diversity will create problems that all parties will need to address and work out on a continuing basis. Examples of areas where this problem resolution may be required include tool standardization; data exchange; version management; electronic communication; data rights; documentation uniformity; configuration management; error identification, analysis, and elimination; product ownership; component integration; and testing.

Lesson 3: *The need to interface with other language programs may constrain the type of Ada features that can be used.* The Ada language design run-time concept does not map directly to the hard real-time environment within the MK2 system. Therefore, attempts must be made to overlay the Ada model on top of the inherited real-time operating system, which has necessitated eliminating certain Ada features (e.g., tasking). Other Ada features not used include generics, dynamic allocation, and full-range data typing. Performance also has suffered, and portability has been minimized. The need to interface with other language programs may result in a loss of the advantages of strong Ada typing and may affect debugging, testing, certification, and the like.

Lesson 4: *To ensure programming uniformity, a style guide should be developed and used across all developer teams.* Use of a common style guide will enhance overall

maintainability of developed code. It also will help control Ada feature utilization, and the code can be automatically checked by applying a pre-process tool. The use of a "pretty printer" post-processing mechanism for human-readable outputs could also enhance software maintainability.

I.10 P-3C UPDATE IV Ada DEVELOPMENT

The objective of the P-3C UPDATE IV Program is to develop a fully integrated, distributive bus, data processing system with improved mission avionics systems. The full weapon system is to be tailored for both retrofit into P-3C predecessor aircraft and forward fit into successor Maritime Patrol Aircraft. The program successfully progressed through the D&V phase between November 1984 and April 1987. After the Milestone II decision in July 1987, Boeing was awarded an FFP contract for FSD to develop, fabricate, qualify vendors, install the system into a P-3C platform, and conduct vendor flight tests by July 1990. The schedule called for Government testing of the flying test bed between July 1990 and February 1992 with subsequent approval for full production to be granted in April 1992.

The program includes the distributive bus data processing Distributed Processor/Display Generator Unit (DP/DGU) system, which consists of six Motorola 68020-based processor modules/DGUs tied together via a dual 1553B bus architecture. Major mission systems avionics include the AN/UYS-2 acoustic processor, the Motorola 68020 based AN/ALR-66(V) 5 Electronic Support Measures (ESM) system, and the AN/APS-137 (V) 3 Inverse Synthetic Aperture Radar. The data processing system and ESM are CFE, and the acoustic processor and the radar are GFE.

The program has been delayed by both hardware and software development difficulties. The current schedule calls for Boeing to deliver the flying test bed to the Government between October 1992 and February 1993.

As one of the first large Ada developments (over 1 million SLOC), the P-3C Update IV program has been a pioneer in the use of Ada. Boeing personnel have made several correct choices in developing software in a new programming language for which the software development environment was immature or limited. First, Boeing's choice of using the VERDIX VADS was a good one. VERDIX has been a leader in the development of Ada software engineering tools, and VADS was one of the best Ada software development environments available at the time of program initiation. Equally good was the choice of the Ready Systems kernel as the core for the operating system. Finally, Boeing's naming convention for Top Level and Lower Level Computer Software Components (TLCSCs/LLCSCs), packages, units, and identifiers has also been beneficial. The naming convention has been very useful in

tracing requirements to design and code and is helpful when reading the PDL and computer source code.

The paragraphs below summarize the lessons learned about Ada use in this program.

Lesson 1: *The Ada code requires more up-front time and effort, and the learning curve is slower.* The software size and development schedule estimates were understated by all parties during the initial phase of the program. The table below lists SLOC estimates at program initiation, at completion of PDR, and in August, 1991.

Computer Software Configuration Item (CSCI)	Best and Final Offer (BAFO) (April 1987)	June 1988	August 1991
DP/DGU	383,530	468,654	565,431
Minimum Mode Software (MMS)	0	37,900	52,764
Electronic Support Measure (ESM)	52,340	58,000	55,516
Acoustic Interface Unit (AIU)	68,900	68,900	97,371
AN/UYS-2	37,320	38,000	147,500
System Avionics Integration Laboratory (SAIL)	218,600	172,870	211,766
Integration Test Software (ITS)	5,000	96,900	109,359
Software Development Laboratory (SDL) (Boeing-Developed Code Only)	37,500	45,500	62,800
TOTAL	803,190	986,724	1,302,507

The final SLOC total should exceed August 1990 estimates by over 10% before completion of software development. The initial sizing estimates will be in error by approximately 100% at program completion.

The Boeing estimates for the software development schedule were predicated on available non-Ada HOL usage. Individual task estimates were too short and did not anticipate the increased up-front work in Ada design and coding that was needed. This fact and a slower than anticipated learning curve for coders resulted in a realized progress rate of 85% of plan for coding, test, and integration activities.

Lesson 2: *Increased facilities and memory are required to accommodate Ada code.* The physical number of hardware tools was initially insufficient to support a software development of this magnitude. This lack of hardware capacity was experienced in all areas of the software development environment, from the SUN workstations used during initial code and testing to the SAIL used for system integration. More SUN workstations were needed to avoid bottlenecks in coding, both in the SDL and at the subvendor locations involved in tactics and correlation programming efforts. The Boeing SDL grew from two SUN 3/280 server stations with 33 SUN client workstations in the fall of 1987 to five SUN 3/280 server stations with 45 SUN client workstations in the fall of 1990.

The SDL mainframes used for the target hardware software build process could not construct a software build in an acceptable period. Initial software program builds took up to 1 week to compile and link. The SDL initially contained one VAX 11/785, one VAX 11/750, and two VAX 8700s. To accommodate the software development demands, the SDL was upgraded by the fall of 1990 to include one VAX 11/785, one VAX 11/750, two VAX 6000/440s, and one VAX 8600. Disk storage capacity was also increased to approximately 40 gigabytes. This increase in hardware capacity has reduced system build time to approximately 8 hours.

Initial plans called for target integration to be conducted on a single SAIL that contained as much actual UPDATE IV hardware as possible, including the full DP/DGU system. A DON SAIL was held at Boeing instead of being delivered to DON to accommodate the integration overload impact on the Boeing SAIL.

Lesson 3: *Some software development tools are immature and have not been proven for many applications.* Immaturity and/or unavailability of software development tools also complicated early software development efforts. A comprehensive Ada support environment was unavailable for early development work. Available tools were immature and were not integrated into a comprehensive package. In addition, available tools were very resource intensive, which exacerbated the previously mentioned hardware problems.

The SUN workstation software build installations initially required 1 week and contained numerous errors because of excessive operator intervention. Upgraded SUN station software and software tool/automation development resulted in eventual turnaround times of 1 day. Error reduction was excellent as a result of the automated tools.

The initial SDL VAX systems were plagued with software and hardware faults, which resulted in numerous system crashes and an average down-time of 1/2 day per week. By applying pressure to Digital Equipment Corporation, fixes were put in place over a period of 1 year, which resulted in mature, stable system performance.

The VERDIX compiler was selected for use after screening available compilers by the procedures recommended in 1987. However, numerous early software and hardware errors were encountered before stable performance was achieved. As late as January 1991, the VERDIX Ada compiler with the version 6.0 program was found to have an optimizer error. After the compiler is corrected, the UPDATE IV program will require a total recompile to remove inefficiencies scattered throughout.

Lesson 4: *Tailoring of DOD software development standards must be addressed to accommodate Ada-unique capabilities.* Although DOD-STD-2167A does not require that software development efforts follow the traditional waterfall model associated with DOD software developments, it does not provide guidance on alternatives. Ada forces more detailed design earlier in the software developments than do previous languages because of the Ada package specifications and the strong data types imposed by Ada. These factors encourage a pseudo "rapid prototyping" approach rather than the traditional waterfall during the design phases.

DOD-STD-2167A does not address distributed processor systems or multiple configuration item developments. Ada was designed specifically for a modular approach to large software developments. For example, DOD-STD-2167A does not adequately address testing between multiple configuration items or the integration phase issues. DOD-STD-2167A documentation neither reflects Ada terminology or structure nor addresses an appropriate approach to documentation development.

Lesson 5: *Although the adoption of Ada was envisioned to enhance the software development process, use of Ada does not guarantee sound software engineering practice.* Specific areas where Ada does not substitute for sound engineering practices include:

- Establishment of system and software requirements—A requirements analysis phase must be conducted to produce appropriate system-level requirements that are then allocated to hardware/software as appropriate. Participation by

both contractor and Government system engineering personnel throughout this evolution is critical to program success.

- Enforcement of control points—The contract must require and the Government must enforce a variety of control points. These control points must take into account Ada-unique development approaches where the approach differs from the traditional DOD-STD-2167A waterfall model. Allowing the contractor to proceed past these control points, even if he does so "at risk," imposes significant risk on successful program completion.

- Configuration management—Use of Ada does not preclude Government requirements for establishing and controlling the functional, allocated, and product baselines. Use of Ada may complicate control of the software allocated baseline by inviting inclusion of design detail into the software requirements documents. Although Ada forces more detailed design earlier in the software development process, the temptation to include this detail into the software allocated baseline must be avoided.

- Testing—The mapping of Ada constructs to DOD-STD-2167A "units," "modules," and "system" is imprecise and can lead to inadequate testing of Ada code. The DOD-STD-2167A premise of fully qualifying a software entity at one level of abstraction before combining that entity into larger integrated components should be maintained. A software entity should not be considered fully qualified solely because the higher level entity into which it is incorporated successfully passes its qualification requirements.

Lesson 6: ***Strict configuration management and control are required to enforce discipline to counter complexity-induced confusion.*** Lack of familiarity with Ada, a slow learning curve for new coders, and schedule delays re-emphasize the absolute requirement to maintain strict software and hardware control within all facilities. With differing levels of coding, unit and package testing, informal integration testing, and formal systems testing occurring in the respective facilities, strict configuration management within the facilities and within the software development library was mandated. Initial SDFs were audited by the Government and found deficient. Replication of numerous informal tests could not be accomplished from the SDFs, as written.

I.11 STANDARD FINANCIAL SYSTEM REDESIGN

The Standard Financial System (STANFINS) is part of the total U.S. Army accounting system and serves as a field-level system for general funds servicing posts, camps, and stations. The original STANFINS was a batch processing system written

in COBOL. A STANFINS Redesign project (STANFINS-R) was undertaken to overhaul the system and make it interactive.

STANFINS-R consists of two subsystems—Subsystems I and II—to be developed independently. This large system is designed to handle mainstream accounting applications such as the general ledger, accounts receivable, fixed assets, and cost accounting standards. The system consists of 500 programs with approximately 2 million lines of code, and it generates 147 reports. The contract for developing Subsystem II, which originally was viewed as a large COBOL project, was awarded to the Computer Sciences Corporation (CSC) in the fall of 1986. However, the contract was modified in the spring of 1988, and Ada was designated as the development language. The first pilot teams were formed in the spring of 1988, and the actual writing of code and Ada bindings began in the fall of 1988. Software development testing and software qualification testing started in the summer and fall of 1989, respectively. Most of the system tests have been completed, and part of the project is operational.

The project was developed in an automated program support environment composed of six Rational R-1000 machines. The code was eventually ported to the target environment, an IBM mainframe running OS/VMS.

Despite delays in the implementation schedule and budget overruns, the STANFINS-R project indicates that there are several advantages to using Ada in information systems development. For example, programmer productivity has been quite high (594 lines of code per staff month), almost double that of typical COBOL projects, and the quality of the software, as evidenced by the test results, appears to be uniformly high.

In many ways, STANFINS-R is a prototypical information systems project from which many lessons, including those described below, can be learned about Ada use.

Lesson 1: ***The available pool of developers skilled in Ada is limited.*** When making projections about project costs, the issue of the limited number of available personnel skilled in Ada and the need for training should be considered. STANFINS-R originally was conceived as a COBOL project. When denial of the waiver resulted in a switch to Ada as the development language, it became apparent that the available pool of developers with Ada/MIS experience was small. The existing staff of COBOL programmers had to be trained in Ada, which caused delay in project execution.

Lesson 2: ***Few compilers and support tools are available for information systems development in the IBM environment that use Ada.*** STANFINS-R demonstrated that

Ada is a viable language for developing information systems in environments where COBOL has been the dominant development language. However, the IBM environment, which is the primary environment for developing such systems, is poorly supported in terms of compilers and support tools. STANFINS-R was the first Ada application of its kind and size to be developed to run on an IBM OS/VMS environment. Because of the lack of available tools to support Ada in this environment, a set of support tools, such as code generators and screen painters, had to be developed as part of the project. Moreover, the compiler, which was developed by Intermetrics but had not been validated, did not provide support for a Configuration Item-based teleprocessing monitor; therefore, the contractor had to write one. In addition, the Database Management System (DBMS) package chosen for the project (i.e., Datacom DB) did not contain a suitable Ada interface; therefore, a hook had to be written. For Ada to be a feasible language for use in developing information systems, the issue of availability of compilers and support tools must be addressed. Most Ada vendors do not offer products in this environment. The dominant compiler in this environment (offered by Intermetrics) has not been validated. The unavailability of suitable compilers has been a significant factor inhibiting the use of Ada in information systems and has created an adverse cycle of events. On the one hand, because Ada is not the preferred language in information system development, vendors have little incentive to offer products to work on the platforms on which such applications are traditionally developed. On the other hand, the paucity of suitable products works to limit the consideration of Ada in developing information systems. Successful implementation of the mandate to use Ada will require a suitable resolution of this cycle. A plausible way to address this problem would be to create appropriate incentive structures that will encourage vendors to develop such products.

Lesson 3: ***Systems developed in Ada may be more maintainable than those written in COBOL.*** Although it is too early to state definitively that Ada maintenance requirements are lower than those for COBOL, preliminary evidence indicates this may be so. Part of STANFINS is operational and has a maintenance staff of six programmers, a much smaller team than would be required to maintain a system of similar size that uses COBOL.

Lesson 4: ***In principle, portability is ensured by developing code in Ada; however, in practice, portability is limited.*** Porting the code from the Rational environment to the target environment was problematic. For a variety of reasons, parts of the code that worked well on the Rational environment did not work in the target environment. For example, nested generics would not work on the target although they tested and compiled on the development machine. Executable code sharing could not be implemented on the target, thereby causing the executable sizes to grow to unmanageable proportions. Other features, such as representation specifications,

Unchecked_Conversion, and Pragma Inline, were not implemented in the target compiler.

Developers in this project had significant problems with porting code from the development environment to the target environment. What compiled on the Rational R-1000 also compiled on the IBM mainframe using OS/MVS. However, lack of compiler support for the teleprocessing monitor and interfaces to the DBMS necessitated the creation of low-level functionally limited code, thereby limiting portability to other environments without significant modifications. Thus, while most of the code can be ported to a VAX/VMS environment, for example, complete portability would require significant alterations.

Lesson 5: *Ada has special advantages that make reuse more feasible and enables the benefits of reuse to be realized at several levels.* At one level are specific packages and templates that can be used in other parts of the project or in other projects. While the same could, in principle, be accomplished with code written in COBOL, the use of generics and packages gives Ada a special advantage over COBOL that makes such reuse much more feasible. There was significant use of these templates at STANFINS. The issue of reuse can also be thought of in terms of tools that are developed for specific projects but with suitable modifications can be used in other projects. The STANFINS project, for example, entailed the development of PSL/PSA tools for writing design specifications. These tools can plausibly be modified and reused in other projects. A follow-up project, the Standard Army Financial Accounting and Reporting System (STARFIARS), demonstrates reusability at both levels. While STARFIARS is likely to be at least 33% larger than STANFINS, the project is scheduled to take 50% less time than STANFINS. The rationale for this aggressive schedule is twofold: STANFINS provided a useful learning curve from which STARFIARS will benefit, and more importantly, the implementation of STANFINS has created system templates and tools that can be reused to create the new system with greater productivity.

Lesson 6: *Current documentation standards need to be reexamined.* The documentation required for STANFINS-R, which was prepared according to the requirements mandated in AIS DOD-STD- 7935-A, was inordinately large. While the exact figure is difficult to ascertain, a conservative estimate is that every line of code generated at least ten lines of documentation. The voluminous documentation clearly limits its usefulness and points to the need to reexamine current documentation standards.

I.12 RECONFIGURABLE MISSION COMPUTER PROJECT

The Reconfigurable Mission Computer (RMC) Project sought to demonstrate that modularity in both hardware and software would reduce the cost of developing new

or upgrading existing embedded systems. The thrust of the project was to exploit hardware and software commonality in different embedded systems.

Lesson 1: *For small technology demonstration projects, anticipate a lack of Ada compilers for small, embedded computers that use advanced microprocessors.* Primary constraints on missile general purpose data processors are size, power, and cycles per second. There is always a drive to use the most advanced microprocessors available to get as much performance as possible in as small a space as possible while consuming the least power possible. Ada compiler vendors, however, are not going to market a compiler until they can determine that it is financially realistic to do so. Small technology demonstrations that want to use Ada in the software development may be restricted to using processors for which a commercial Ada compiler exists.

Lesson 2: *Plan on allocating a portion of the Central Processing Unit (CPU) utilization to the inefficiencies of using a modular design approach and design implementation in Ada.* The RMC project goals included creating portable Ada programs, running them on several platforms, measuring the code change required, and learning what it took to make an Ada program portable. A modular design based on Abstract Data Types was used to hide machine interfaces. We also hid the "goodies" the compiler vendors offered outside of the Ada language behind our own package interfaces. The results were a reduction in performance that can be made up with a higher throughput CPU. However, any throughput increase realized by upgrading platforms is usually given to the analyst to develop more capable algorithms. A modular design in Ada can reduce code, test, and modification times and is well worth the extra overhead incurred.

Lesson 3: *Plan on throwing away some or all of the first software designed.* After the first design and implementation of demonstration software in Ada, it was felt that the implementation would be improved the next time. Fortunately, we had the luxury of doing just that, and we understood and implemented a much better software system the second time. It is not necessary to wait until all of the tools and hardware are in place to begin coding. As much of the design as possible should be implemented as soon as possible. A commercial prototype or similar system should be used to gain understanding of the system, and the first cut should be used to verify that the requirements can be met.

Lesson 4: *Dedicate an individual or group (depending on the size of the project) to the Ada-hardware interface.* Ada touches the "iron" in several places: the target debug monitor for on-target program development; the kernel for time, memory, and processor management; and device drivers used by the application. A person or group needs to be familiar with hardware registers, ports, memory locations, and the

low-level facilities available in Ada. Evolving hardware architectures and compiler upgrades make this an absolute necessity.

I.13 INTELLIGENT MISSILE PROJECT

The purpose of this project, which is funded by the Office of Naval Technology under the Missile Support Technology block NW2A, is to develop generic software techniques and to design tools that will allow the use of knowledge-based artificial intelligence (AI) paradigms for control and decision-making functions in missiles. These capabilities are to be implemented in Ada. Having these features will yield more adaptive and autonomous missile operation.

A simple, forward-chaining inference engine was developed and tested on several computers. Next, a decision-tree type of expert system was developed along with a tool (in Ada) to generate the Ada decision tree. (None of the commercial expert system shells could do this at the time.) Finally, a hybrid system was developed that combined the flexibility of an inference engine with the speed of a decision tree. Execution performance was measured for all three types of systems. The decision tree was the fastest, and the hybrid system was a close second.

Lesson 1: *System analysis must be conducted at the beginning to ensure that adequate resources (e.g., compilers, hardware platform) will be available for meeting the system requirements. The tendency to favor particular hardware or compiler systems just because they are available must be avoided.* Choosing a particular CPU simply because it is available can lead to problems that could have been avoided by performing adequate system analysis. One problem encountered was that the CPU needed an assembly language program to be downloaded and run to "kickstart" the CPU so that Ada code could be downloaded and executed. The CPU was hardwired to have a certain memory configuration that was incompatible with where the Ada code had to be.

Similarly, using a compiler that already is on hand without ensuring it can do the job also will lead to delays. In this case, the compiler had been validated for a particular single-board computer. Although the vendor stated that it should work with the chosen target board, the vendor would not provide any help because compilers for other CPUs had a much higher priority, although the highest level of maintenance available had been purchased for this project.

Lesson 2: *Although Ada has many features, it does not have everything from every language. Ada is restricted in the type of AI systems for which it can be used.* Because it is a procedural language, Ada has some restrictions, particularly with regard to certain AI applications. In LISP, an arbitrary string of characters can be handled in three different ways: as test, as a variable, or as a function to be called. This ability,

which is very useful for building production-type expert systems, results from LISP being not only a language but also an environment. Because Ada is only a language, there are restrictions on the types of production expert systems that can be implemented. Although it would be possible to implement the equivalent LISP environment in Ada, LISP is too slow and big for a missile system, which was the reason for its not being used in the first place.

Appendix J
FY91 Ada Technology Insertion Program Projects

This appendix provides a brief description of the Ada Technology Insertion Program (ATIP) projects funded in FY91. The projects fall into three primary categories—education, bindings, and technology. For more information on these projects, contact the Ada Joint Program Office at (703) 614-0209.

J.1 EDUCATION

Of the 14 projects funded, one addresses Ada education.

Undergraduate Curriculum and Course Development in Software
Defense Advanced Research Projects Agency (DARPA)

This program will support the development of educational materials using Ada that will be widely distributed to and used by educators; will enhance the software engineering content of courses and course sequences in computer science curricula; and will demonstrate, through pilot implementations, the feasibility and viability of a comprehensive undergraduate curriculum in software engineering using Ada.

J.2 BINDINGS

The eight bindings projects are grouped into the following categories:

- Government Open Systems Interconnection Profile (GOSIP)
- Management Information System (MIS) Mathematical Binding
- Military Standard (MIL-STD)-1553
- Portable Operating System Interface for UNIX (POSIX)
- Structured Query Language (SQL)
- XWindows

Ada Application Program Interface to GOSIP Network Services
Defense Information Systems Agency (DISA) (Formerly DCA)

GOSIP is a family of protocols that supports network services. Although Ada bindings to GOSIP exist, this project will develop a robust Ada/GOSIP binding for standardizing the interface of Ada applications to GOSIP network services.

Decimal Arithmetic
U.S. Air Force

Compiler vendors support decimal arithmetic but in nonstandard ways. This project will standardize a mechanism for realizing COBOL-style exact decimal arithmetic in Ada 83. It will provide sufficient functionality to handle financial applications with at least 18 digits of precision. It will offer early availability with Ada 83 compilers, notational convenience, ease of transition to Ada 9X, and run-time efficiency.

Generic Avionics Data Bus Toolkit
U.S. Navy

This project will offer a standard software interface that can be reused for various MIL-STD multiplex data buses with minimal changes. The initial software will focus on the MIL-STD-1553B protocol because this protocol is the most prevalent, but it will be designed to be configured for expansion to other types of data buses. An integrated MIL-STD-1553B monitor with debugging tools is planned.

POSIX/Ada Real-time Bindings
U.S. Air Force/Navy

POSIX defines a collection of system services that provide portable application interfaces to operating systems. The POSIX effort is divided into several areas that cover the range of operating system services. These include basic system services, real-time services, security services, user command interface, user graphical interface, network services, mail services, and system administration. This project will develop draft Ada bindings for the real-time service area (POSIX 1003.4 and 1003.4a standards), work with the Institute of Electrical and Electronics Engineers (IEEE) standards organization to promote the use of these drafts as a starting point for development of standard Ada bindings, and develop a test prototype implementation of Ada tasking using the 1003.4 (real-time) and 1003.4a (threads) services.

Ada SQL Interface Standardization
Defense Advanced Research Projects Agency (DARPA)

SQL is a set of standards associated with relational databases and data dictionaries. The SQL Ada Module Description Language (SAMeDL) provides an interface technology for Ada applications accessing SQL database management systems. The ATIP program will fully document both the SAMeDL as a language and its supporting methodology, respond to the needs of the standardization process, and

coordinate efforts of potential vendors of SAMeDL processors as well as identify needs of potential SAMeDL customers to assist the transition to the SAMeDL.

A SAMeDL Pilot Project on SIDPERS-3
U.S. Air Force/Army

A SAMeDL tool set will be developed consisting of a SAMeDL Module Manager and a SAMeDL compiler. These tools will target a designated database running on an Everex Personal Computer (PC) under UNIX. Both an existing application and a new application will be developed using this tool set. This effort is designed to prove that the SAMeDL tool set has the robustness, maturity, and potential for reusability to be employed as the Ada/SQL binding of choice on any large Department of Defense (DOD), Ada Management Information System (MIS) program.

Common Ada XWindow Interface (CAXI)
U.S. Navy

XWindows is a *de facto* industry standard that provides a graphical user interface. Popular toolkit extensions to XWindows include Open Look (used by AT&T, SUN, and others) and Open Software Foundation MOTIF (used by IBM, Digital Equipment Corporation, Hewlett Packard, Apollo, and others). This project will design and produce a common interface to both the Open Look and MOTIF toolkits. The interface will be written in Ada and will allow application programs to use either toolkit without modification to the application program. This will increase the portability of Ada applications and provide flexibility in the selection of hardware.

An Interactive Ada/XWindows User Interface Generator
U.S. Army

This project proposes to develop a general-purpose Ada/XWindows User Interface Generator that automatically generates Ada source code. Using this tool, a developer will be able to interactively develop a functioning user interface by selecting user interface primitives and arranging them on the screen. This tool is intended to reduce the bottleneck imposed upon Ada systems developers when developing window-based user interfaces based on the XWindows system and the MOTIF toolkit.

J.3 TECHNOLOGY

The five projects in the technology category deal with the following:

- Engineering environments
- Prototyping
- Reuse
- Security

AdaSAGE Enhancements

U.S. Air Force/Army/Navy

AdaSAGE is an applications development set of utilities designed to facilitate rapid and professional construction of systems in Ada. AdaSAGE was developed by the Department of Energy at the Idaho National Engineering Laboratory. Applications may vary from small to large multiprogram systems using special capabilities. These capabilities include database storage and retrieval (SQL compliant), graphics, communications, formatted windows, on-line help, sorting, and editing. AdaSAGE operates on various systems including MS-DOS platforms, UNIX System V, and OS/2. A developer using the Ada language and the AdaSAGE development system can design a product tailored to a specific requirement that offers outstanding performance and flexibility. The ATIP proposal provides enhancements to AdaSAGE requested by the user community and supports the creation of a computer-aided training program.

ATLAS/Ada-Based Enhancements for Test (ABET)

U.S. Air Force

ABET is an Air Force and IEEE effort to provide an international standard for an automatic test environment for maintenance activities. Ada is the language to be used for implementing this standard. ABET will intelligently incorporate Ada into the test arena by providing a set of layered standards to the test community.

A Computer-Aided Prototyping System for Real-Time Software

U.S. Air Force

The program will demonstrate a high-technology, low-cost approach to providing state-of-the-art software prototyping tools for real-time Ada programs. It provides the opportunity to use the thesis efforts of students at the Naval Postgraduate School, who are DOD personnel familiar with Ada and its embedded applications.

Reusable Ada Products for Information Systems Development (RAPID)
U.S. Army

RAPID is an Ada reuse program that includes an automated library tool for configuration, identification, and retrieval of reusable Ada software components and a staff that supports and trains developers in reusability and sound software engineering principles. Its mission is to ensure that the DOD objective of reusable, maintainable, and reliable Ada software is achieved. It provides a total reuse program supporting the entire software development life.

Ada Reuse in a Trusted Message Processing System for Real-Time Software
U.S. Navy

This project will investigate Ada reuse in developing software that satisfies the Orange Book B2-Level security requirement. The system will be fielded as the Submarine Message Buffer (SMB) System, supporting personnel with two levels of security clearance.

Appendix K
Navy and Marine Corps Ada Projects

A database of Navy and Marine Corps projects that use Ada has been assembled for reference by Program Managers who are planning to use or currently are using Ada. The database includes the following information:

- Project Name
- Project Description
- Application Area, (i.e., C2, C3, C4I, EW, Space, Communication, Armament, Ordnance, Acoustic, Navigation, Financial, Personnel, Contracting, Material Management, Medical, Depot Maintenance, Tool, DBMS, Graphical, Education, Simulation, Other)
- Sponsor/Developer
- Point of Contact and Phone Number
- Program Status (in planning, developed, completed, or canceled)
- Source Lines of Code
- Host System
- Target System

Because this database is very large, its contents have not been included in this version of the *Ada Implementation Guide*. This database is available either on disk as a Lotus 1-2-3 file or in hard copy. To obtain a copy, please fill out the attached order form.

If you would like your project to be considered for inclusion in this database, please provide the information listed on the order form.

ORDER FORM

Program Name _____
Prog. Manager _____
Address _____
City, St & Zip _____

Please send:

_____ (1) Copy of DON Ada Projects Database on Disk (LOTUS 1-2-3 File) and/or

_____ (1) Hard copy of DON Ada Projects Database

. . . . . .

To have your project considered for inclusion in this database, please provide the following information:

- Project Name
- Project Description (brief & concise)
- Application Area, (i.e., C2, C3, C4I, EW, Space, Communication, Armament, Ordnance, Acoustic, Navigation, Financial, Personnel, Contracting, Material Management, Medical, Depot Maintenance, Tool, DBMS, Graphical, Education, Simulation, Other)
- Sponsor/Developer
- Point of Contact and Phone Number
- Program Status (in planning, developed, completed, or canceled)
- Source Lines of Code
- Host System
- Target System

Please send this order form and/or project information to:

Space & Naval Warfare Systems Command
SPAWAR 2241 (CDR M. Romeo)
2451 Crystal Drive (CPK-5, 700)
Washington, DC 20363-5100

Appendix L
GLOSSARY

ABET	Ada-Based Enhancements for Test
ACEC	Ada Compiler Evaluation Capability
ACM	Association for Computing Machinery
ACVC	Ada Compiler Validation Capability
AdaIC	Ada Information Clearinghouse
AdaJUG	Ada Joint (Services) Users Group
Ada PSE	Ada Programming Support Environment
ADP	Automatic Data Processing
AES	Ada Evaluation System
AFATDS	Advanced Field Artillery Tactical Data System
AFB	Air Force Base
AFSC	Air Force Systems Command
AI	Artificial Intelligence
AIE	Ada Integrated Environment
AIS	Automated Information System
AIU	Acoustic Interface Unit
AJPO	Ada Joint Program Office
ALS	Ada Language System
ALS/N	Ada Language System/Navy
AMMWS	Advanced Millimeter Wave Seeker
AMPS	Advanced Message Processing System
ANSI	American National Standards Institute
AP	Acquisition Plan
APP	Application Portability Profile
ASEET	Ada Software Engineering Education and Training
ASIS	Ada Semantic Interface Specification
ASP	Acquisition Strategy Plan
ASR	Ada Software Repository
AST	Advanced Systems Technology
ASW	Anti-Submarine Warfare
ASWSOW	Anti-Submarine Standoff Weapon
AT&T	American Telephone & Telegraph
ATCCS	Army Tactical Command and Control System
ATF	Advanced Tactical Fighter
ATIP	Ada Technology Insertion Program
ATRIM	Aviation Training and Readiness System
AVF	Ada Validation Facilities

BAFO	Best and Final Offer
BBS	Bulletin Board System
BP	Backplane
C3I	Command, Control, Communications, and Intelligence
C4I	Command, Control, Communications, Computers, and Intelligence
CAB	Common Ada Baseline
CAIS	Common Ada PSE Interface Set
CALS	Computer-Aided Logistics Support
CAMP	Common Ada Missile Packages
CASE	Computer-Aided Software Engineering
CAS REPS	Casualty Reporting System
CAUWG	Commercial Ada Users Working Group
CAXI	Common Ada XWindow Interface
CC&I	Command, Control, and Intelligence
CCITT	International Consultative Committee for Telegraph and Telephone
CCP	Code Counting Program
CCS	Combat Control System
CDA	Central Design Agency
CDIF	CASE Data Interchange Format
CDPA	Central Design Programming Activity
CDR	Critical Design Review
CDRL	Contract Data Requirements List
CECOM	Communications Electronics Command
CERT/CC	Computer Emergency Response Team Coordination Center
CFE	Contractor-Furnished Equipment
CGI	Computer Graphics Interface
CI	Configuration Item
CIF	Central Issue Facility
CIM	Corporate Information Management
CLNP	Connectionless Network Protocol
CLOC	Compiled/Assembled Lines of Code
CMM	Capability Maturity Model
CMS-2	Compiler Monitor System-2
CMU	Carnegie-Mellon University
CMU/SEI	Carnegie-Mellon University/Software Engineering Institute
COBOL	Common Business Oriented Language
COE	Common Operating Environment

COMNAVCOMTELCOM	Commander, Naval Computer and Telecommunications Command
COMSPAWARSYSCOM	Commander, Space and Naval Warfare Systems Command
COTS	Commercial Off-The-Shelf
CPDL	Computer Program Development Laboratory
CPU	Central Processing Unit
CREASE	Catalog of Resources for Education
CRISD	Computer Resource Integrated Software Document
CRLCMP	Computer Resources Life-Cycle Management Plan
CRSS	C3I Reusable Software System
CSC	Computer Sciences Corporation
CSCI	Computer Software Configuration Item
CSS	Centralized Structure Store
CSS	Computer Sciences School
CSU	Computer Software Unit
CWG	Coordinator Working Group
D&V	Demonstration & Validation
DAB	Defense Acquisition Board
DACS	Data and Analysis Center for Software
DARPA	Defense Advanced Research Projects Agency
DBMS	Database Management System
DC	Device Coordinate
DCDS	Distributed Computing Design System
DDI	Directorate of Defense Information
DDN	Defense Data Network
DDR&E	Director of Defense Research and Engineering
DE1	Data Elements in the Source
DEMVAL	Demonstration Evaluation
DFCS	Digital Flight Control System
DID	Data Item Description
DISA	Defense Information Systems Agency
DOD	Department of Defense
DON	Department of the Navy
DP/DGU	Distributed Processor/Display Generator Unit
DSRS	Defense Software Repository System
DTC II	Desk Top Computer II
DTIC	Defense Technical Information Center
DWS	Defense Weapons System

ECLD	Embedded Comment Lines in Data
ECLS	Embedded Comment Lines in Source
ECM	Electronic Countermeasures
ECMA	European Computer Manufacturing Association
EDL	Event-Driven Language
EMR	Extended Memory Reach
ENB	Engineering Notebook
EPROM	Erasable Programmable Read Only Memory
EP	Enhanced Processor
ERA	Entity Relationship Attribute
ESD	Electronic Systems Division
ESM	Electronic Support Measure
FAR	Federal Acquisition Regulations
FAU	Fin Actuator Unit
FCDSSA	Fleet Combat Direction System Support Activity
FFP	Firm Fixed Price
FFRDC	Federally Funded Research and Development Center
FIPS	Federal Information Processing Standards
FRAWG	Front Range Ada Working Group
FSD	Full-Scale Development
FTAM	File Transfer, Access, and Management
ftp	File Transfer Protocol
43RSS	AN/UYK-43 Run-Time Support System
GAO	General Accounting Office
GEU	Guidance Electronics Unit
GFE	Government-Furnished Equipment
GFS	Government-Furnished Software
GKS	Graphical Kernel System
GNCP	Guidance, Navigation, and Control Program
GOSIP	Government Open Systems Interconnection Profile
GPEF	Generic Package of Elementary Functions
GPPF	Generic Package of Primitive Functions
GPO	Government Printing Office
GRACE™	Generic Reusable Ada Com ıents for Engineering
GSIS	Graphics System Interface Standard
GTRIMS	Ground Controller Training System
HOL	High Order Language
HPBP	High-Performance Backplane
HPP	High-Performance Processor

IBM	International Business Machines
ICE	In-Circuit Emulator
IEC	International Electro-Technical (Committee)
IEEE	Institute of Electrical and Electronics Engineers
IGES	Initial Graphics Exchange Specification
ILSP	Integrated Logistic Support Plan
IMU	Inertial Measurement Unit
INEL	Idaho National Engineering Laboratory
InProc	In Processing
I/O	Input/Output
IPR	In-Process Review
IPS	Integrated Project Summary
IPSE	Integrated Project Support Environment
IRAC	International Requirements and Design Criteria
IRDS	Information Resource Dictionary System
IRS	Interface Requirements Specification
ISA	Instruction Set Architecture
ISDN	Integrated Services Digital Network
ISEA	In-Service Engineering Activity
ISEE	Integrated Software Engineering Environment
ISO	International Standards Organization
ISSC	Information System Software Center
ITS	Integrated Test Software
IV&V	Independent Verification and Validation
JCS	Joint Chiefs of Staff
JTC	Joint Technical Committee
KAPSE	Kernel Ada Programming Support Environment
LAN	Local Area Network
LCM	Life-Cycle Maintenance
LCSA	Life-Cycle Support Activity
MAPSE	Minimal Ada Programming Support Environment
MCCDC	Marine Corps Combat Development Command
MCCR	Mission-Critical Computer Resources
MCCRES	Marine Corps Combat Readiness Evaluation System
MENS	Mission Element Need Statement
MEPS	Message Edit Processing System
MHS	Message Handling Service
MIL-HDBK	Military Handbook

MIL-STD	Military Standard
MIMMS	Marine Corps Integrated Maintenance Management System
MIS	Management Information System
MMS	Minimum Mode Software
MOA	Memorandum of Agreement
MOTS	Military Off-The-Shelf
NAC	Naval Avionics Center
NADC	Naval Air Development Center
NARDAC	Navy Regional Data Automation Center
NASA	National Aeronautics and Space Administration
NASEE	NAVAIR Software Engineering Environment
NATO	North Atlantic Treaty Organization
NAUG	Navy Ada Users Group
NAVAIR	Naval Air Systems Command
NAVDAC	Navy Data Automation Command
NAVSEA	Naval Sea Systems Command
NAVSWC	Naval Surface Warfare Center
NCS	Network Computing Service
NCTAMS	Naval Computer and Telecommunications Area Master Station
NCTAMS LANT	NCTAMS Atlantic
NCTAMS EASTPAC	NCTAMS Eastern Pacific
NCTC	Naval Computer and Telecommunications Command
NCTS	Naval Computer and Telecommunications Station
NDC	Normalized Device Coordinate
NDI	Nondevelopmental Item
NGCR	Next Generation Computer Resources
NISBS	NATO Interoperable Submarine Broadcast System
NIST	National Institute of Standards and Technology
NISMC	Naval Information System Management Center
NOSC	Naval Ocean Systems Center
NSWC	Naval Surface Weapons Center
NTIS	National Technical Information Service
NUSC	Naval Undersea Command
NWRC	Navy Wide Reuse Center
NWSUS	Navy WWMCCS Site-Unique Software
OAS	Offensive Avionics System
OASD	Office of the Assistant Secretary of Defense
OCD	Operational Concept Document

OFPS	Operational Flight Program Size
OMU	Operational Mock-up
OOD	Object-Oriented Design
OORA	Object-Oriented Requirements Analysis
OPE	Open Systems Environment
OPNAVINST	Naval Operations Instruction
OPR	Office of Primary Responsibility
ORG	Organization Chain of Command
OS	Operating System
OSA	Open Systems Architecture
OSE	Open Systems Environment
OSF	Open Software Foundation
OSI	Open Systems Interconnection
OSS	Operations Support System
OSSWG	Operating Systems Standards Working Group
PC	Personal Computer
PCIS	Portable Common Interface Set
PCTE	Portable Common Tool Environment
PDL	Program Design Language
PDR	Preliminary Design Review
PDS	Post-Deployment Support
PDSS	Post-Deployment Software Support
PDU	Pulse Driver Unit
PHIGS	Programmers Hierarchical Interactive Graphics System
PIWG	Performance Issues Working Group
PMC	Project Management Charter
POSIX	Portable Operating System Interface for UNIX
PPBS	Planning, Programming, and Budgeting System
PRR	Product Readiness Review
PSE	Project (or Programming) Support Environment
PSESWG	Project Support Environment Standard Working Group
R&D	Research and Development
RACS	Registration and Access Control System
RADC	Requirements and Design Criteria
RAM	Random Access Memory
RAPID	Reusable Ada Products for Information Systems Development
RCL	RAPID Center Library
RDA	Remote Database Access

RDBMS	Relational Database Management System
RDT&E	Research, Development, Test, and Evaluation
RES	Resources
RFP	Request for Proposals
RLF	Reuse Library Framework
RMC	Reconfigurable Mission Computer
ROM	Read Only Memory
RPC	Remote Process Communication
RSC	Reusable Ada Software Component
RTE	Run-Time Environment
SAE	Software Architectures Engineering
SAFENET	Survivable Adaptable Fiber-optic Embedded Network
SAI	Software Action Item
SAIL	System Avionics Integration Laboratory
SAME	SQL Ada Module Extension
SAMeDL	SQL Ada Module Description Language
SASSY	Supported Activities Supply System
SCCS	Submarine Combat Control System
SCE	Software Capability Evaluation
SCL	Stand-alone Comment Lines
SCMP	System Configuration Management Plan
SCS	Submarine Combat System
SDC-W	Software Development Center, Washington
SDD	System Design Definition
SDE	Software Development Environment
SDF	Software Development Folder
SDIO	Strategic Defense Initiative Organization
SDL	Software Development Laboratory
SDP	Software Development Plan
SDP	System Division Paper
SDR	System Design Review
SDSR	Software Development Status Report
SECNAVINST	Secretary of the Navy Instruction
SECR	Standard Embedded Computer Resources
SEE	Software Engineering Environment
SEI	Software Engineering Institute
SEMP	System Engineering Management Plan
SEO	Software Executive Official
SEPG	Software Engineering Process Group
SGS/AC	Shipboard Gridlock System with Auto-Correlation
SGML	Standard Generalized Markup Language

SIGAda	Special Interest Group on Ada
SIGSOFT	Special Interest Group on Software Engineering
SIL	System Integration Laboratory
SIP	System Integration Plan
SLOC	Source Lines of Code
SLOCWC	Source Lines of Code Without Comments
SMB	Submarine Message Buffer
SMM	Software Management Metrics
SMP	Software Master Plan
SOW	Statement of Work
SPA	Software Process Assessment
SPAWAR	Space and Naval Warfare Systems Command
SPC	Software Productivity Consortium
SPR	Software Problem Report
SQAP	Software Quality Assurance Plan
SQL	Structured Query Language
SRC	Software Requirements Change
SRR	System Requirements Review
SRS	Software Requirements Specification
SSA	Software Support Activity
STANFINS	Standard Financial System
STARFIARS	Standard Army Financial Accounting and Reporting System
STARS	Software Technology for Adaptable, Reliable Systems
STSC	Software Technology Support Center
SUP	Support Planning
SWAP	Software Action Plan
SWG	Special Working Group
SWTP	Software Technology Plan
TACAMO	Take Charge and Move Out
TACFIRE	Tactical Fire Direction
TADSTAND	Tactical Digital Standard
TC	Target Capacity
TCL	Total Comment Lines
TCP/IP	Transmission Control Protocol/Internet Protocol
TDA	Technical Directive Authority
TDT	Theater Display Terminal
TEMP	Test and Evaluation Master Plan
TEP	Test and Evaluation Plan
TFA	Transparent File Access
TLCSC/LLCSC	Top Level/Lower Level Computer Software Component

TLOC	Total Lines of Code
TOES	Telephone Order-Entry System
TSGCEE	Tri-Service Group on Communications and Electronics Equipment
UIMS	User Interface Management System
ULLS	Unit Level Logistics System
USMC	U.S. Marine Corps
USW	Undersea Warfare
VADS	Verdix Ada Development System
VDI	Virtual Device Interface
VHSIC	Very High-Speed Integrated Circuit
VRC	Virtual Reference Coordinate
VSR	Validation Summary Reports
VT	Virtual Terminal
WAdaS	Washington Ada Symposium
WAM	WWMCCS ADP Modernization
WBS	Work Breakdown Structure
WC	World Coordinate
WIS	WWMCCS Information System
WWMCCS	World Wide Military Command and Control System